Algorithms for Reasoning with graphical models

Slides Set 10:
Bounded Inference Non-iteratively;
Mini-Bucket Elimination

Rina Dechter

(Class Notes (8-9), Darwiche chapter 14)
Outline

• Mini-bucket elimination
• Weighted Mini-bucket
• Mini-clustering
• Re-parameterization, cost-shifting
• Iterative Belief propagation
• Iterative-join-graph propagation
Types of queries

<table>
<thead>
<tr>
<th>Type</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max-Inference</td>
<td>[ f(x^*) = \max_x \prod_{\alpha} f_\alpha(x_\alpha) ]</td>
</tr>
<tr>
<td>Sum-Inference</td>
<td>[ Z = \sum_x \prod_{\alpha} f_\alpha(x_\alpha) ]</td>
</tr>
<tr>
<td>Mixed-Inference</td>
<td>[ f(x_M^*) = \max_{x_M} \sum_{x_S} \prod_{\alpha} f_\alpha(x_\alpha) ]</td>
</tr>
</tbody>
</table>

- **NP-hard**: exponentially many terms
- We will focus on **approximation** algorithms
  - **Anytime**: very fast & very approximate! Slower & more accurate
Queries

• Probability of evidence (or partition function)

\[ P(e) = \sum_{X-\text{var}(e)} \prod_{i=1}^{n} P(x_i \mid pa_i) \mid_e \]
\[ Z = \sum_{X} \prod_{i} \psi_i(C_i) \]

• Posterior marginal (beliefs):

\[ P(x_i \mid e) = \frac{P(x_i, e)}{P(e)} = \frac{\sum_{X-\text{var}(e) \setminus X_i} \prod_{j=1}^{n} P(x_j \mid pa_j) \mid_e}{\sum_{X-\text{var}(e)} \prod_{j=1}^{n} P(x_j \mid pa_j) \mid_e} \]

• Most Probable Explanation

\[ \bar{x}^* = \arg\max_{\bar{x}} P(\bar{x}, e) \]
Bucket Elimination

Query: \( P(a \mid e = 0) \propto P(a, e = 0) \)

Elimination Order: d, e, b, c

\[
P(a, e = 0) = \sum_{c, b, e = 0} P(a)P(b \mid a)P(c \mid a)P(d \mid a, b)P(e \mid b, c)
\]

\[
= P(a)\sum_c P(c \mid a)\sum_b P(b \mid a)\sum_{e = 0} P(e \mid b, c)\sum_d P(d \mid a, b)
\]

Original Functions

- D: \( P(d \mid a, b) \)
- E: \( P(e \mid b, c) \)
- B: \( P(b \mid a) \)
- C: \( P(c \mid a) \)
- A: \( P(a) \)

Messages

- \( f_D(a, b) = \sum_d P(d \mid a, b) \)
- \( f_E(b, c) = P(e = 0 \mid b, c) \)
- \( f_B(a, c) = \sum_b P(b \mid a) f_D(a, b) f_E(b, c) \)
- \( f_C(a) = \sum_c P(c \mid a) f_B(a, c) \)

\[
P(a, e = 0) = p(A) f_C(a)
\]

Bucket Tree

- D, A, B
- E, B, C
- B, A, C
- C, A
- A

Time and space \( \exp(w^*) \)
Finding MPE/MAP

Algorithm BE-mpe (Dechter 1996, Bertele and Briochi, 1977)

\[ \text{MPE} = \max_{a,e,d,c,b} p(a) p(c|a) p(b|a) p(d|b,a) p(e|b,c) \]

- **bucket B**: 
  \[ p(b|a) \ p(d|b,a) \ p(e|b,c) \]

- **bucket C**: 
  \[ p(c|a) \lambda_{B \rightarrow C}(a,d,c,e) \]

- **bucket D**: 
  \[ \lambda_{C \rightarrow D}(a,d,e) \]

- **bucket E**: 
  \[ \mathbf{1}[e = 0] \lambda_{D \rightarrow E}(a,e) \]

- **bucket A**: 
  \[ p(a) \lambda_{E \rightarrow A}(a) \]

\[ \text{OPT} \]

\[ W^* = 4 \text{ “induced width” (max clique size)} \]
Generating the Optimal Assignment

- Given BE messages, select optimum config in reverse order

\[
\begin{align*}
    b^* &= \arg \max_b p(b|a^*) p(d^*|b, a^*) p(e^*|b, c^*) \\
    c^* &= \arg \max_c p(c|a^*) \lambda_{B\rightarrow C}(a^*, c, d^*, e^*) \\
    d^* &= \arg \max_d \lambda_{C\rightarrow D}(a^*, d, e^*) \\
    e^* &= \arg \max_e \mathbb{1}[e = 0] \lambda_{D\rightarrow E}(a^*, e) \\
    a^* &= \arg \max_a p(a) \cdot \lambda_{E\rightarrow A}(a)
\end{align*}
\]

Return optimal configuration \((a^*, b^*, c^*, d^*, e^*)\)

\[
\begin{align*}
    B: & & p(b|a) \ p(d|b, a) \ p(e|b, c) \\
    C: & & p(c|a) \ \lambda_{B\rightarrow C}(a, c, d, e) \\
    D: & & \lambda_{C\rightarrow D}(a, d, e) \\
    E: & & \mathbb{1}[e = 0] \ \lambda_{D\rightarrow E}(a, e) \\
    A: & & p(a) \ \lambda_{E\rightarrow A}(a)
\end{align*}
\]

\(\text{OPT} = \text{optimal value}\)
Approximate Inference

• Metrics of evaluation

• **Absolute error**: given \( e > 0 \) and a query \( p = P(x|e) \), an estimate \( r \) has absolute error \( e \) iff \( |p-r| < e \)

• **Relative error**: the ratio \( r/p \) in \([1-e,1+e]\).

• Dagum and Luby 1993: approximation up to a relative error is NP-hard.

• Absolute error is also NP-hard if error is less than .5
Outline

• Mini-bucket elimination
• Weighted Mini-bucket
• Mini-clustering
• Re-parameterization, cost-shifting
• Iterative Belief propagation
• Iterative-join-graph propagation
Mini-Buckets: “Local Inference”

• Computation in a bucket is time and space exponential in the number of variables involved

• Therefore, partition functions in a bucket into “mini-buckets” on smaller number of variables
Decomposition Bounds

- Upper & lower bounds via approximate problem decomposition
- Example: MAP inference \( F(x) = f_1(x) + f_2(x) \)

\[
\begin{array}{c|c}
  x & F(x) \\
  \hline
  0 & 1.0 \\
  1 & 4.0 \\
  2 & 6.0 \\
  3 & 0.0 \\
\end{array}
\quad = \quad
\begin{array}{c|c}
  x & f_1(x) \\
  \hline
  0 & 1.0 \\
  1 & 2.0 \\
  2 & 3.0 \\
  3 & 4.0 \\
\end{array}
\quad \times \quad
\begin{array}{c|c}
  x & f_2(x) \\
  \hline
  0 & 1.0 \\
  1 & 2.0 \\
  2 & 2.0 \\
  3 & 0.0 \\
\end{array}
\]

\[
\max_x F(x) \quad = \quad \max_x \left[ f_1(x) \times f_2(x) \right]
\]

\[
4.0 \quad \leq \quad \left[ \max_x f_1(x) \times \max_x f_2(x) \right] \quad = \quad 4.0 \times 2.0 \quad = \quad 8.0
\]

- Relaxation: two “copies” of \( x \), no longer required to be equal
- Bound is tight (equality) if \( f_1, f_2 \) agree on maximizing value \( x \)
Mini-Bucket Approximation

Split a bucket into mini-buckets → bound complexity

\[
\text{bucket } (X) = \left\{ f_1, f_2, \ldots f_r, f_{r+1}, \ldots f_n \right\}
\]

\[
\lambda_X(\cdot) = \max_x \prod_{i=1}^n f_i(x, \ldots)
\]

\[
\lambda_{X,1}(\cdot) = \max_x \prod_{i=1}^r f_i(x, \ldots)
\]

\[
\lambda_{X,2}(\cdot) = \max_x \prod_{i=r+1}^n f_i(x, \ldots)
\]

\[
\lambda_X(\cdot) \leq \lambda_{X,1}(\cdot) \lambda_{X,2}(\cdot)
\]

Exponential complexity decrease: \(O(e^n) \rightarrow O(e^r) + O(e^{n-r})\)
Mini-Bucket Elimination

\[ \lambda_{B \rightarrow D}(a, d) = \max_b P(d \mid a, b) \cdot p(b \mid a) \]

\[ \lambda_{B \rightarrow C}(e, c) = \max_b P(e \mid b, c) \]

\[ \lambda_{B \rightarrow D}(a, d) = \max_d \ldots \]

\[ U = \text{upper bound} \]

[Dechter & Rish 2003]
Mini-Bucket Elimination

\[ \lambda_{B\to D}(a, d) = \max_b P(d|a, b) p(b|a) \]
\[ \lambda_{B\to C}(e, c) = \max_b P(e|b, c) \]
\[ \lambda_{B\to D}(a, d) = \max_d \text{...} \]

\( U = \text{upper bound} \)

[Dechter & Rish 2003]

\[ P(d|a, b) p(b|a) \]
\[ P(e|b', c) \]
\[ P(c|a) \]
\[ \lambda_{B\to C}(e, c) \]
\[ \lambda_{B\to D}(a, d) \]
\[ \lambda_{C\to E}(a, e) e=0 \]
\[ \lambda_{D\to A}(a) \]

\[ \lambda_{E\to A}(a) \]

\[ \lambda_{D\to A}(a) \]
Mini-Bucket Elimination

[Dechter and Rish, 1997; 2003]
Mini-Bucket Decoding

• Assign values in reverse order using approximate messages

\[ b^* = \arg \max_b P(e^*|b, c^*)P(d|a^*, b) \ p(b|a^*) \]
\[ c^* = \arg \max_e \lambda_{B \rightarrow C}(e^*, c) \]
\[ d^* = \arg \max_d \lambda_{B \rightarrow D}(a^*, d) \]
\[ e^* = 0 \]
\[ a^* = \arg \max_a P(a) \ \lambda_{E \rightarrow A}(a) \ \lambda_{D \rightarrow A}(a) \]

Greedy configuration = lower bound

\[ U = \text{upper bound} \]

\[ \text{return}(a^*, e^*, d^*, c^*, b^*) \]
Semantics of Mini-Bucket: Splitting a Node

Variables in different buckets are renamed and duplicated
(Kask et. al., 2001), (Geffner et. al., 2007), (Choi, Chavira, Darwiche, 2007)

Before Splitting:
Network $N$

After Splitting:
Network $N'$
**MBE-MPE(i): Algorithm MBE-mpe**

- **Input**: \( I \) – max number of variables allowed in a mini-bucket
- **Output**: [lower bound (\( P \) of suboptimal solution), upper bound]

**Example: MBE-mpe(3) versus BE-mpe**

\[ U = \text{Upper bound} \quad w^* = 2 \]

\[ \text{OPT} \quad w^* = 4 \]
Mini-Bucket Decoding (for min-sum)

\[ \hat{b} = \arg \min_b f(\hat{a}, b) + f(b, \hat{c}) \]
\[ + f(b, \hat{d}) + f(b, \hat{e}) \]
\[ \hat{c} = \arg \min_c \lambda_{B \rightarrow C}(\hat{a}, c) + f(c, \hat{a}) + f(c, \hat{e}) \]
\[ \hat{d} = \arg \min_d f(\hat{a}, d) + \lambda_{B \rightarrow D}(d, \hat{e}) \]
\[ \hat{e} = \arg \min_e \lambda_{C \rightarrow E}(\hat{a}, e) + \lambda_{D \rightarrow E}(\hat{a}, e) \]
\[ \hat{a} = \arg \min_a f(a) + \lambda_{E \rightarrow A}(a) \]

Greedy configuration = upper bound

L = lower bound

[Dechter and Rish, 2003]
(i,m)-Partitionings

Definition 7.1.1 ((i,m)-partitioning) Let \( H \) be a collection of functions \( h_1, \ldots, h_t \) defined on scopes \( S_1, \ldots, S_t \), respectively. We say that a function \( f \) is subsumed by a function \( h \) if any argument of \( f \) is also an argument of \( h \). A partitioning of \( h_1, \ldots, h_t \) is canonical if any function \( f \) subsumed by another function is placed into the bucket of one of those subsuming functions. A partitioning \( Q \) into mini-buckets is an \((i,m)\)-partitioning if and only if (1) it is canonical, (2) at most \( m \) non-subsumed functions are included in each mini-bucket, (3) the total number of variables in a mini-bucket does not exceed \( i \), and (4) the partitioning is refinement-maximal, namely, there is no other \((i,m)\)-partitioning that it refines.
MBE(i,m), MBE(i)

• Input: Belief network \(( P_1, \ldots, P_n )\)
• Output: upper and lower bounds
• Initialize: put functions in buckets along ordering
• Process each bucket from \( p=n \) to 1
  • Create \((i,m)\)-partitions
  • Process each mini-bucket
• (For mpe): assign values in ordering \( d \)
• Return: mpe-configuration, upper and lower bounds
Algorithm MBE-mpe(i,m)

Input: A belief network \( \mathcal{B} = \langle X, D, G, P_G, \Pi \rangle \), where \( P = \{ P_1, \ldots, P_n \} \); an ordering of the variables, \( d = X_1, \ldots, X_n \); observations \( \omega \).

Output: An upper bound \( U \) and a lower bound \( L \) on the most probable configuration given the evidence. A suboptimal solution \( x^* \) that provides the lower bound \( L = P(x^*) \).

1. Initialize: Generate an ordered partition of the conditional probability function, \( \text{bucket}_1, \ldots, \text{bucket}_n \), where \( \text{bucket}_i \) contains all functions whose highest variable is \( X_i \). Put each observed variable in its bucket.

2. Backward: For \( p \leftarrow n \) downto \( 1 \), do
   for all the functions \( h_1, h_2, \ldots, h_j \) in \( \text{bucket}_p \), do
   
   - If (observed variable) \( \text{bucket}_p \) contains \( X_p = x_p \), assign \( x_p = x_p \) to each function and put each in appropriate bucket.
   
   - else, generate an \( (i,m) \)-partitioning, \( Q' = \{ Q_1, \ldots, Q_p \} \) of \( h_1, h_2, \ldots, h_j \) in \( \text{bucket}_p \).
   
   - for each \( Q_1 \in Q' \) containing \( h_1, \ldots, h_j \), do
     
     \[
     h_t \leftarrow \max_{X_t} \prod_{j=1}^{i} h_j
     \]  

     Add \( h_t \) to the bucket of the largest-index in \( \text{scope}(h_t) \). Put constants in \( \text{bucket}_1 \).

3. Forward:

   - Compute an mpe cost by maximizing over \( X_1 \), the product in \( \text{bucket}_1 \). Namely \( U \leftarrow \max_{X_1} \prod_{h_j \in \text{bucket}_1} h_j \).
   
   - (Generate an approximate mpe tuple): Given \( x_{(1 \ldots (i-1))} = (x_1, \ldots, x_{i-1}) \) choose \( x_i = \arg\max_{X_i} \prod_{h_j \in \text{bucket}_1} h_j (x_{1 \ldots (i-1)}) \). \( L \leftarrow P(x_1, \ldots, x_n) \)

4. Output \( U \) and \( L \) and configuration: \( \hat{x} = (x_1, \ldots, x_n) \).
Partitioning, Refinements

Clearly, as the mini-buckets get smaller, both complexity and accuracy decrease.

Definition 7.1.4 Given two partitionings $Q'$ and $Q''$ over the same set of elements, $Q'$ is a refinement of $Q''$ if and only if for every set $A \in Q'$ there exists a set $B \in Q''$ such that $A \subseteq B$.

It is easy to see that:

Proposition 7.1.5 If $Q''$ is a refinement of $Q'$ in bucket$_p$, then $h_p \leq g_{Q'}^p \leq g_{Q''}^p$. 
Properties of MBE(i)

- **Complexity**: $O(r \exp(i))$ time and $O(\exp(i))$ space
- Yields a lower bound and an upper bound
- **Accuracy**: determined by upper/lower (U/L) bound
- Possible use of mini-bucket approximations
  - As *anytime algorithms*
  - As *heuristics* in search
- Other tasks (similar mini-bucket approximations)
  - Belief updating, Marginal MAP, MEU, WCSP, Max-CSP
    [Dechter and Rish, 1997], [Liu and Ihler, 2011], [Liu and Ihler, 2013]
Anytime Approximation

Algorithm anytime-mpe($\epsilon$)
Input: Initial values of $i$ and $m$, $i_0$ and $m_0$; increments $i_{\text{step}}$ and $m_{\text{step}}$, and desired approximation error $\epsilon$.
Output: $U$ and $L$

1. **Initialize:** $i = i_0$, $m = m_0$.
2. **do**
3. run mbe-mpc($i,m$)
4. $U \leftarrow$ upper bound of mbe-mpc($i,m$)
5. $L \leftarrow$ lower bound of mbe-mpc($i,m$)
6. Retain best bounds $U$, $L$, and best solution found so far
7. **if** $1 \leq U/L \leq 1 + \epsilon$, **return** solution
8. **else** increase $i$ and $m$: $i \leftarrow i + i_{\text{step}}$ and $m \leftarrow m + m_{\text{step}}$
9. **while** computational resources are available
10. **Return** the largest $L$
    and the smallest $U$ found so far.
MBE for Belief Updating and for Probability of Evidence or Partition Function

• Idea mini-bucket is the same:

\[
\sum_{x} f(x) \cdot g(x) \leq \sum_{x} f(x) \cdot \sum_{x} g(x)
\]

\[
\sum_{x} f(x) \cdot g(x) \leq \sum_{x} f(x) \cdot \max_{x} g(X)
\]

• So we can apply a sum in each mini-bucket, or better, one sum and the rest max, or min (for lower-bound)

• \text{MBE-bel-max}(i,m), \text{MBE-bel-min}(i,m) generating upper and lower-bound on beliefs approximates \text{BE-bel}

• \text{MBE-map}(i,m): max mini-buckets will be maximized, sum mini-buckets will be sum-max. Approximates \text{BE-map}.
Algorithm MBE-bel-max(i,m)

Input: A belief network $\mathcal{B} = \langle X, D, P_G, \prod \rangle$, an ordering $d = (X_1, \ldots, X_n)$; evidence $e$

Output: an upper bound on $P(X_1, e)$ and an upper bound on $P(e)$.

1. Initialize: Partition $P = \{P_1, \ldots, P_n\}$ into buckets $bucket_1, \ldots, bucket_n$, where $bucket_k$ contains all CPTs $h_1, h_2, \ldots, h_t$ whose highest-index variable is $X_k$.

2. Backward: for $k = n$ to 2 do
   
   - If $X_p$ is observed ($X_k = a$), assign $X_k \leftarrow a$ in each $h_j$ and put the result in the highest-variable bucket of its scope (put constants in $bucket_1$).
   
   - Else for $h_1, h_2, \ldots, h_t$ in $bucket_k$ Generate an $(i,m)$-partitioning, $Q' = \{Q_1, \ldots, Q_t\}$. For each $Q_i \in Q'$, containing $h_{l_1}, \ldots, h_{l_t}$, do
     
     $$h_l \leftarrow \sum_{X_k} \prod_{j=1}^{l-1} h_{1,j}, \text{ if } l = 1$$
     
     $$h_l \leftarrow \max_{X_k} \prod_{j=1}^{l-1} h_{1,j}, \text{ if } k \neq 1$$

     Add $h_l$ to the bucket of the highest-index variable in its scope $\bigcup_{j=1}^{l} scope(h_{1,j}) - \{X_k\}$. (put constant functions in $bucket_1$).

3. Return $P'(x_1, e) \leftarrow$ the product of functions in the bucket of $X_1$, which is an upper bound on $P(x_1, e)$.

   $P'(e) \leftarrow \sum_{x_1} P'(\bar{x}_1, e)$, which is an upper bound on probability of evidence.

Figure 8.5: Algorithm MBE-bel-max(i,m).
CPCS Networks – Medical Diagnosis (noisy-OR model)

Test case: no evidence

Anytime-mpe(0.0001)
U/L error vs time

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>cpcs360</th>
<th>cpcs422</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>elim-mpe</strong></td>
<td>115.8</td>
<td>1697.6</td>
</tr>
<tr>
<td><strong>anytime-mpe(ε), ε = 10^{-4}</strong></td>
<td>70.3</td>
<td>505.2</td>
</tr>
<tr>
<td><strong>anytime-mpe(ε), ε = 10^{-1}</strong></td>
<td>70.3</td>
<td>110.5</td>
</tr>
</tbody>
</table>
Outline

- Mini-bucket elimination
- Weighted Mini-bucket
- Mini-clustering
- Re-parameterization, cost-shifting
- Iterative Belief propagation
- Iterative-join-graph propagation
Decomposition for Sum

\[ \sum_{x} f_1(x) \cdot f_2(x) \leq \left[ \sum_{x} f_1(x)^{\frac{1}{w_1}} \right]^{w_1} \cdot \left[ \sum_{x} f_2(x)^{\frac{1}{w_2}} \right]^{w_2} \]

- Generalize technique to sum via Holder’s inequality:

\[ \sum_{x_1} f(x_1) = \left[ \sum_{x_1} f(x_1)^{\frac{1}{w_1}} \right]^{w_1} \]

- Define the weighted (or powered) sum:

\[ \sum_{x_1} \sum_{x_2} w_1 \cdot w_2 \neq \sum_{x_2} \sum_{x_1} w_2 \cdot w_1 \]

- “Temperature” interpolates between sum & max:

\[ \lim_{w \to 0^+} \sum_{x} w f(x) = \max_{x} f(x) \]

- Different weights do not commute:
The Power Sum and Holder Inequality

The power sum is defined as follows:

\[ \sum_{x}^{w} f(x) = \left( \sum_{x} f(x)^{\frac{1}{w}} \right)^{w} \]  

(1.2)

where \( w \) is a non-negative weight. The power sum reduce to a standard summation when \( w = 1 \) and approaches max when \( w \to 0^+ \).

[Holder inequality] Let \( f_i(x), i = 1..r \) be a set of functions and \( w_1, ..., w_r \) be a set of non-zero weights, s.t., \( w = \sum_{i=1}^{r} w_i \) then,

\[ \sum_{x}^{w} \prod_{i=1}^{r} f_i(x) \leq \prod_{i=1}^{r} \sum_{x}^{w_i} f_i(x) \]
Working Example

• Model:
  • Markov network

• Task:
  • Partition function

\[
Z = \sum_{A,B,C} f(A)f(B)f(C)f(A,B)f(A,C)f(B,C)
\]
Mini-Bucket (Basic Principles)

• Upper bound

\[ \sum_{i} a_i b_i \leq (\sum_{i} a_i) \max(b_i) \]

• Lower bound

\[ \sum_{i} a_i b_i \geq (\sum_{i} a_i) \min(b_i) \]

I am using \( a_i b_i \) to represent the general constant.
Holder Inequality

\[ \sum_{i} a_i b_i \leq \left( \sum_{i} a_i^{1/w_1} \right)^{w_1} \left( \sum_{i} b_i^{1/w_2} \right)^{w_2} \]

• Where \(a_i > 0, b_i > 0\) and \(w_1 + w_2 = 1, w_1 > 0, w_2 > 0\)

• When \(\frac{a_i^{1/w_1}}{\sum a_i^{1/w_1}} = \frac{b_i^{1/w_2}}{\sum b_i^{1/w_2}}\), the equality is achieved.

(Qiang Liu slides)

Reverse Holder Inequality

- If \( w_1 + w_2 = 1 \), but \( w_1 < 0, w_2 > 1 \) the direction of the inequality reverses.

\[
\sum_i a_i b_i \geq \left( \sum_i a_i^{1/w_1} \right)^{w_1} \left( \sum_i b_i^{1/w_2} \right)^{w_2}
\]

(Qiang Liu slides)

Weighted Mini-Bucket

(for summation)

Exact bucket elimination:
\[ \lambda_B(a, c, d, e) = \sum_b [f(a, b) \cdot f(b, c) \cdot f(b, d) \cdot f(b, e)] \]
\[ \leq \left[ \sum_b f(a, b)f(b, c) \right] \cdot \left[ \sum_b f(b, d)f(b, e) \right] \]
\[ = \lambda_{B\rightarrow C}(a, c) \cdot \lambda_{B\rightarrow D}(d, e) \]

where \( \sum_x f(x) = \left[ \sum_x f(x)^{1/w} \right]^w \)

is the weighted or "power" sum operator

\[ \sum_x f_1(x)f_2(x) \leq \left[ \sum_x f_1(x) \right] \left[ \sum_x f_2(x) \right] \]

where \( w_1 + w_2 = w \) and \( w_1 > 0, w_2 > 0 \)

(lower bound if \( w_1 > 0, w_2 < 0 \))

\[ U = \text{upper bound} \]

[slides10 828X 2019]
Algorithm Weighted WMBE(i,m), \((w_1, \ldots, w_n)\)

Input: A belief network \(\mathcal{B} = (\mathcal{X}, \mathcal{D}, P_{\mathcal{G}}, \Pi)\), an ordering \(d = (X_1, \ldots, X_n)\); evidence \(e\)

Output: an upper bound on \(\sum_{X} \prod_{i=1}^{n} P_i\)

1. **Initialize**: Partition \(P = \{P_1, \ldots, P_n\}\) into buckets \(\text{bucket}_1, \ldots, \text{bucket}_n\), where \(\text{bucket}_k\) contains all CPTs \(h_1, h_2, \ldots, h_t\) whose highest-index variable is \(X_k\).

2. **Backward**: for \(k = n\) to 1 do
   - If \(X_p\) is observed \((X_k = a)\), assign \(X_k \leftarrow a\) in each \(h_j\) and put the result in the highest-variable bucket of its scope (put constants in \(\text{bucket}_1\)).
   - Else for \(h_1, h_2, \ldots, h_t\) in \(\text{bucket}_k\) Generate an \((i,m)\)-partitioning, \(Q' = \{Q_1, \ldots, Q_r\}\). Select a set of weights \(w_1, \ldots, w_r\) s.t \(\sum w_i = w\). For each \(Q_t \in Q'\), containing \(h_{t_1}, \ldots, h_{t_t}\), do
     \[
     h_t \leftarrow \sum_{X_k} \prod_{j=1}^{i} h_{t_j} = \left(\sum_{X_k} \prod_{j=1}^{t} (h_{t_j})^{w_j}\right)\frac{1}{w_t}
     \]
     Add \(h_t\) to the bucket of the highest-index variable in its scope (and put constant functions in \(\text{bucket}_1\)).

3. **Return** \(U\) \(\leftarrow\) the weighted product of functions in the bucket of \(X_1\), which is an upper bound on \(P(x_1, e)\).
Weighted-mini-bucket for Marginal Map
Bucket Elimination for MMAP

MAP* is the marginal MAP value

\[ \lambda(A, C, D, E) \]

\[ \lambda^E(A) \]

\[ \lambda^D(A, E) \]

\[ \lambda^C(A, D, E) f(A, C) f(C, E) \]

\[ \sum_C \]

\[ \lambda^B(A, C, D, E) f(A, C) f(B, C) f(B, D) f(B, E) \]

\[ \sum_B \]

\[ \max_D \]

\[ \max_E \]

\[ \max_D \]

\[ \lambda^D(A, E) f(A, D) \]

\[ \lambda^C(A, D, E) f(A, C) f(C, E) \]

\[ \lambda^B(A, C, D, E) f(A, C) f(B, C) f(B, D) f(B, E) \]

\[ \sum_B \]

\[ \max_D \]

\[ \lambda^D(A, E) f(A, D) \]

\[ \lambda^C(A, D, E) f(A, C) f(C, E) \]

\[ \lambda^B(A, C, D, E) f(A, C) f(B, C) f(B, D) f(B, E) \]

\[ \sum_B \]

\[ \max_D \]

\[ \max_E \]

\[ \max_E \]

\[ \max_D \]

\[ \lambda^D(A, E) f(A, D) \]

\[ \lambda^C(A, D, E) f(A, C) f(C, E) \]

\[ \lambda^B(A, C, D, E) f(A, C) f(B, C) f(B, D) f(B, E) \]

\[ \sum_B \]

\[ \max_D \]

\[ \max_E \]

\[ \max_D \]

\[ \lambda^D(A, E) f(A, D) \]

\[ \lambda^C(A, D, E) f(A, C) f(C, E) \]

\[ \lambda^B(A, C, D, E) f(A, C) f(B, C) f(B, D) f(B, E) \]

\[ \sum_B \]

\[ \max_D \]

\[ \max_E \]

\[ \max_D \]

\[ \lambda^D(A, E) f(A, D) \]

\[ \lambda^C(A, D, E) f(A, C) f(C, E) \]

\[ \lambda^B(A, C, D, E) f(A, C) f(B, C) f(B, D) f(B, E) \]

\[ \sum_B \]

\[ \max_D \]

\[ \max_E \]

\[ \max_D \]

\[ \lambda^D(A, E) f(A, D) \]

\[ \lambda^C(A, D, E) f(A, C) f(C, E) \]

\[ \lambda^B(A, C, D, E) f(A, C) f(B, C) f(B, D) f(B, E) \]

\[ \sum_B \]

\[ \max_D \]

\[ \max_E \]

\[ \max_D \]

\[ \lambda^D(A, E) f(A, D) \]

\[ \lambda^C(A, D, E) f(A, C) f(C, E) \]

\[ \lambda^B(A, C, D, E) f(A, C) f(B, C) f(B, D) f(B, E) \]

\[ \sum_B \]

\[ \max_D \]

\[ \max_E \]

\[ \max_D \]

\[ \lambda^D(A, E) f(A, D) \]

\[ \lambda^C(A, D, E) f(A, C) f(C, E) \]

\[ \lambda^B(A, C, D, E) f(A, C) f(B, C) f(B, D) f(B, E) \]

\[ \sum_B \]

\[ \max_D \]

\[ \max_E \]

\[ \max_D \]

\[ \lambda^D(A, E) f(A, D) \]

\[ \lambda^C(A, D, E) f(A, C) f(C, E) \]

\[ \lambda^B(A, C, D, E) f(A, C) f(B, C) f(B, D) f(B, E) \]

\[ \sum_B \]

\[ \max_D \]

\[ \max_E \]

\[ \max_D \]

\[ \lambda^D(A, E) f(A, D) \]

\[ \lambda^C(A, D, E) f(A, C) f(C, E) \]

\[ \lambda^B(A, C, D, E) f(A, C) f(B, C) f(B, D) f(B, E) \]

\[ \sum_B \]

\[ \max_D \]

\[ \max_E \]

\[ \max_D \]

\[ \lambda^D(A, E) f(A, D) \]
MB and WMB for Marginal MAP

\[ X_M = \{A, D, E\} \]
\[ X_S = \{B, C\} \]
\[
\max_{X_M} \sum_{X_S} P(X) \quad \lambda_{B \rightarrow C}(a, c) = \sum_b f(a, b) f(b, c) \\
\lambda_{B \rightarrow D}(d, e) = \sum_b f(b, d) f(b, e) \\
\lambda_{E \rightarrow A}(a) = \max_e \lambda_{C \rightarrow E}(a, e) \lambda_{D \rightarrow E}(a, e) \\
U = \max_a f(a) \lambda_{E \rightarrow A}(a)
\]

Marginal MAP

\[ \lambda_{B \rightarrow C}(a, c) = \sum_b f(a, b) f(b, c) \]
\[ \lambda_{B \rightarrow D}(d, e) = \sum_b f(b, d) f(b, e) \]
\[ \lambda_{C \rightarrow E}(a, e) \lambda_{D \rightarrow E}(a, e) \]
\[ f(a) \lambda_{E \rightarrow A}(a) \]
\[ \Sigma_B \quad \Sigma_C \quad \lambda_{B \rightarrow C}(a, c) \quad f(a, c) \quad f(c, e) \]
\[ \lambda_{B \rightarrow D}(d, e) \quad f(a, d) \]
\[ \max_A \quad \max_E \quad \max_D \quad \max_B \]

Can optimize over cost-shifting and weights
(single pass “MM” or iterative message passing)

[U = upper bound]

[Dechter and Rish, 2003]
[Liu and Ihler, 2011; 2013]
MBE-map

Process max buckets
With max mini-buckets
And sum buckets with weighted Mini-buckets
Figure 7.7: Belief network for a linear block code.

Example 7.3.1 We will next demonstrate the mini-bucket approximation for MAP on an example of probabilistic decoding (see Chapter 2). Consider a belief network which describes the decoding of a linear block code, shown in Figure 7.7. In this network, $U_i$ are information bits and $X_i$ are code bits, which are functionally dependent on $U_i$. The vector $(U, X)$, called the channel input, is transmitted through a noisy channel which adds Gaussian noise and results in the channel output vector $Y = (Y^u, Y^e)$. The decoding task is to assess the most likely values for the $U$'s given the observed values $Y = (y^u, y^e)$, which is the MAP task where $U$ is the set of hypothesis variables, and $Y = (y^u, y^e)$ is the evidence. After processing the observed buckets we get the following bucket configuration (lower case $y$'s are observed values):

- $\text{bucket}(X_0) = P(y_0^u | X_0), P(X_0 | U_0, U_1, U_2)$,
- $\text{bucket}(X_1) = P(y_1^u | X_1), P(X_1 | U_1, U_2, U_3)$,
- $\text{bucket}(X_2) = P(y_2^u | X_2), P(X_2 | U_2, U_3, U_4)$,
- $\text{bucket}(X_3) = P(y_3^u | X_3), P(X_3 | U_3, U_4, U_5)$,
- $\text{bucket}(X_4) = P(y_4^u | X_4), P(X_4 | U_5, U_6, U_7)$,
- $\text{bucket}(U_0) = P(U_0), P(y_0^u | U_0)$,
- $\text{bucket}(U_1) = P(U_1), P(y_1^u | U_1)$,
- $\text{bucket}(U_2) = P(U_2), P(y_2^u | U_2)$,
- $\text{bucket}(U_3) = P(U_3), P(y_3^u | U_3)$,
- $\text{bucket}(U_4) = P(U_4), P(y_4^u | U_4)$.

Processing by mini-map($4,1$) of the first top five buckets by summation and the rest by maximization, results in the following mini-bucket partitionings and function generation:
\[ \text{bucket}(X_0) = \{ P(y_0^a | X_0, P(X_0|U_0, U_1, U_2)) \}, \]

\[ \text{bucket}(X_1) = \{ P(y_0^a | X_1, P(X_1|U_1, U_2, U_3)) \}, \]

\[ \text{bucket}(X_2) = \{ P(y_0^a | X_2, P(X_2|U_2, U_3, U_4)) \}, \]

\[ \text{bucket}(X_3) = \{ P(y_0^a | X_3, P(X_3|U_3, U_4, U_0)) \}, \]

\[ \text{bucket}(X_4) = \{ P(y_0^a | X_4, P(X_4|U_4, U_0, U_1)) \}. \]

\[ \text{bucket}(U_0) = \{ P(U_0, P(y_0^a | U_0), h_{X_0}(U_0, U_1, U_2), h_{X_1}(U_3, U_4, U_0)) \}, \]

\[ \text{bucket}(U_1) = \{ P(U_1, P(y_1^a | U_1), h_{X_1}(U_1, U_2, U_3), h_{X_0}(U_4, U_1)) \}, \]

\[ \text{bucket}(U_2) = \{ P(U_2, P(y_2^a | U_2), h_{X_2}(U_2, U_3, U_4), h_{X_3}(U_1, U_1)) \}, \]

\[ \text{bucket}(U_3) = \{ P(U_3, P(y_3^a | U_3), h_{X_3}(U_3, U_4), h_{X_1}(U_4, U_1)) \}, \]

\[ \text{bucket}(U_4) = \{ P(U_4, P(y_4^a | U_4), h_{X_4}(U_4, U_1)) \}. \]

The first five buckets are not partitioned at all and are processed as full buckets, since in this case a full bucket is a \((4,1)\)-partitioning. This processing generates five new functions, three are placed in bucket \(U_0\), one in bucket \(U_1\) and one in bucket \(U_2\). Then bucket \(U_0\) is partitioned into three mini-buckets processed by maximization, creating two functions placed in bucket \(U_1\) and one function placed in bucket \(U_3\). Bucket \(U_1\) is partitioned into two mini-buckets, generating functions placed in bucket \(U_2\) and bucket \(U_5\). Subsequent buckets are processed as full buckets. Note that the scope of recorded functions is bounded by 3.

In the bucket of \(U_4\) we get an upper bound \(U \geq \text{MAP} = P(U, \tilde{y}^a, \bar{y}^a)\) where \(\tilde{y}^a\) and \(\bar{y}^a\) are the observed outputs for the \(U\)'s and the \(X\)'s bits transmitted. In order to bound \(P(U|\bar{e})\), where \(\bar{e} = (\tilde{y}^a, \bar{y}^a)\), we need \(P(\bar{e})\) which is not available. Yet, again, in most cases we are interested in the ratio \(P(U = u_1|\bar{e})/P(U = u_2|\bar{e})\) for competing hypotheses \(U = u_1\) and \(U = u_2\) rather than in the absolute values. Since \(P(U|\bar{e}) = P(U, \bar{e})/P(\bar{e})\) and the probability of the evidence is just a constant factor independent of \(U\), the ratio is equal to \(P(U_1, \bar{e})/P(U_2, \bar{e})\). \(\square\)
Theorem 8.10: Algorithm \text{WMB}(i,m) takes $O(r \cdot k^i)$ time and space, where, $k$ bounds the domain size and $r$ is the number of input functions. For $m = 1$ the algorithm is time and space linear and is bounded by $O(r \cdot \exp(|S|))$, where $|S|$ is the maximum scope of any input function, $|S| \leq i \leq n$. 
Outline

• Mini-bucket elimination
• Weighted Mini-bucket
• **Mini-clustering**
  • Re-parameterization, cost-shifting
  • Iterative Belief propagation
  • Iterative-join-graph propagation
Join-Tree Clustering (Cluster-Tree Elimination)

**EXACT algorithm**

*Time and space: exp(cluster size)= exp(treewidth)*

\[
h_{(1,2)}(b, c) = \sum_a p(a) \cdot p(b \mid a) \cdot p(c \mid a, b)
\]

\[
h_{(2,1)}(b, c) = \sum_{d,f} p(d \mid b) \cdot p(f \mid c, d) \cdot h_{(3,2)}(b, f)
\]

\[
h_{(2,3)}(b, f) = \sum_{c,d} p(d \mid b) \cdot p(f \mid c, d) \cdot h_{(1,2)}(b, c)
\]

\[
h_{(3,2)}(b, f) = \sum_c p(e \mid b, f) \cdot h_{(4,3)}(e, f)
\]

\[
h_{(3,4)}(e, f) = \sum_b p(e \mid b, f) \cdot h_{(2,3)}(b, f)
\]

\[
h_{(4,3)}(e, f) = p(G = g_e \mid e, f)
\]
Mini-Clustering

Split a cluster into mini-clusters \( \implies \) bound complexity

\[
\sum_{\text{elim } i=1}^{n} h_i \leq \left( \sum_{\text{elim } i=1}^{r} h_i \right) \cdot \left( \sum_{\text{elim } i=r+1}^{n} h_i \right)
\]

Exponential complexity decrease \( O(e^n) \rightarrow O(e^{\text{var}(r)}) + O(e^{\text{var}(n-r)}) \)

We can replace the sum with power sum
For weights that sum to 1 in each mini-bucket
Mini-Clustering, i-bound=3

\[ h_{(1,2)}^1(b, c) = \sum_a p(a) \cdot p(b | a) \cdot p(c | a, b) \]

\[ h_{(2,3)}^1(b) = \sum_{c,d} p(d | b) \cdot h_{(1,2)}^1(b, c) \]

\[ h_{(2,3)}^2(f) = \max_{c,d} p(f | c, d) \]

**APPROXIMATE algorithm**

*Time and space: \( \exp(i\text{-}bound) \)*

**Number of variables in a mini-cluster**
Mini-Clustering - Example

\begin{align*}
H_{(1,2)} & \quad h_{(1,2)}^1(b, c) := \sum_a p(a) \cdot p(b \mid a) \cdot p(c \mid a, b) \\
H_{(2,1)} & \quad h_{(2,1)}^1(b) := \sum_{d,f} p(d \mid b) \cdot h_{(3,2)}^1(b, f) \\
H_{(2,1)} & \quad h_{(2,1)}^2(c) := \max_{d,f} p(f \mid c, d) \\
H_{(2,3)} & \quad h_{(2,3)}^1(b) := \sum_{c,d} p(d \mid b) \cdot h_{(1,2)}^1(b, c) \\
H_{(2,3)} & \quad h_{(2,3)}^2(f) := \max_{c,d} p(f \mid c, d) \\
H_{(3,2)} & \quad h_{(3,2)}^1(b, f) := \sum_e p(e \mid b, f) \cdot h_{(4,3)}^1(e, f) \\
H_{(3,4)} & \quad h_{(3,4)}^1(e, f) := \sum_b p(e \mid b, f) \cdot h_{(2,3)}^1(b) \cdot h_{(2,3)}^2(f) \\
H_{(4,3)} & \quad h_{(4,3)}^1(e, f) := p(G = g_e \mid e, f)
\end{align*}
Cluster Tree Elimination vs. Mini-Clustering

CTE

1. ABC
   - $h_{(1,2)}(b, c)$

2. BCDF
   - $h_{(2,1)}(b, c)$
   - $h_{(2,3)}(b, f)$

3. BEF
   - $h_{(3,2)}(b, f)$
   - $h_{(3,4)}(e, f)$

4. EFG
   - $h_{(4,3)}(e, f)$

MC

1. ABC
   - $H_{(1,2)}$ $h_{(1,2)}^1(b, c)$

2. BCDF
   - $H_{(2,1)}$ $h_{(2,1)}^1(b)$ $h_{(2,1)}^2(c)$
   - $H_{(2,3)}$ $h_{(2,3)}^1(b)$ $h_{(2,3)}^2(f)$

3. BEF
   - $H_{(3,2)}$ $h_{(3,2)}^1(b, f)$
   - $H_{(3,4)}$ $h_{(3,4)}^1(e, f)$

4. EFG
   - $H_{(4,3)}$ $h_{(4,3)}^1(e, f)$
Heuristics for partitioning
(Dechter and Rish, 2003, Rollon and Dechter 2010)

**Scope-based Partitioning Heuristic** (SCP) aims at minimizing the number of mini-buckets in the partition by including in each minibucket as many functions as respecting the $i$ bound is satisfied.

Partitioning lattice of bucket \{f_1, f_2, f_3, f_4\}.

- Log relative error:
  \[
  \text{RE}(f, h) = \sum_i (\log(f(t)) - \log(h(t)))
  \]

- Max log relative error:
  \[
  \text{MRE}(f, h) = \max_i \{\log(f(t)) - \log(h(t))\}
  \]

Use greedy heuristic derived from a distance function to decide which functions go into a single mini-bucket.
Greedy Scope-based Partitioning

Procedure **Greedy Partitioning**

**Input:** \( \{h_1, \ldots, h_k\} \), \( i \)-bound;

**Output:** A partitioning \( mb(1), \ldots, mb(p) \) such that every \( mb(i) \) contains at most \( i \)-bound variables;

1. Sort functions by the size of their scopes. Let \( \{h_1, \ldots, h_k\} \) be the sorted array of functions, with \( h_1 \) having the largest scope.
2. for \( i = 1 \) to \( k \)
   - if \( h_i \) can be placed in existing mini-buckets without making the scope greater than the \( i \)-bound, place it in the one with the most functions.
   - else create a new mini-bucket and place \( h_i \) in it.
endfor
Heuristic for Partitioning

**Scope-based Partitioning Heuristic.** The *scope-based* partition heuristic (SCP) aims at minimizing the number of mini-buckets in the partition by including in each mini-bucket as many functions as possible as long as the *i* bound is satisfied. First, single function mini-buckets are decreasingly ordered according to their arity from left to right. Then, each mini-bucket is absorbed into the left-most mini-bucket with whom it can be merged.

The time complexity of Partition\( (B, i) \), where \( B \) is the bucket to be partitioned, and \( |B| \), the number of functions in the bucket, using the SCP heuristic is \( O(|B| \log(|B|) + |B|^2) \).

The scope-based heuristic is quite fast, its shortcoming is that it does not consider the actual information in the functions.
Greedy Partition as a function of a distance function $h$

```
function GreedyPartition(B, i, h)
1. Initialize $Q$ as the bottom partition of $B$;
2. While $\exists Q' \in ch(Q)$ which is a $i$-partition
   $Q \leftarrow \arg\min_{Q'} \{h(Q \rightarrow Q')\}$ among child $i$-partitions of $Q$;
3. Return $Q$;
```

Figure 8.13: Greedy partitioning

**Proposition 8.6.5** The time complexity of GreedyPartition is $O(|B| \times T)$ where $O(T)$ is the time complexity of selecting the min child partition according to $h$. 
Comparing Mini-clustering against Belief Propagation.

What is belief propagation
Iterative Belief Propagation

- Belief propagation is exact for poly-trees
- IBP - applying BP iteratively to cyclic networks

One step: update $\text{BEL}(U_1)$

- No guarantees for convergence
- Works well for many coding networks

slides 10 828X 2019
Linear Block Codes

Input bits

Parity bits

Received bits

Gaussian channel noise

σ
Probabilistic decoding

Error-correcting linear block code

State-of-the-art:
approximate algorithm – iterative belief propagation (IBP)
(Pearl’s poly-tree algorithm applied to loopy networks)
MBE-mpe vs. IBP

MBE-mpe is better on low $w^*$ codes
IBP (or BP) is better on randomly generated (high $w^*$) codes.

Bit error rate (BER) as a function of noise (sigma):

Structured (50,25) block code, $P=7$

Random (100,50) block code, $P=4$
Grid 15x15 - 10 evidence

Grid 15x15, evid=10, w*=22, 10 instances

NHD

<table>
<thead>
<tr>
<th>i-bound</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>NHD</td>
<td>0.00</td>
<td>0.02</td>
<td>0.04</td>
<td>0.06</td>
<td>0.08</td>
<td>0.10</td>
<td>0.12</td>
<td>0.14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MC
IBP

<table>
<thead>
<tr>
<th>Absolute error</th>
<th>0.00</th>
<th>0.01</th>
<th>0.02</th>
<th>0.03</th>
<th>0.04</th>
<th>0.05</th>
<th>0.06</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relative error</th>
<th>0.00</th>
<th>0.02</th>
<th>0.04</th>
<th>0.06</th>
<th>0.08</th>
<th>0.10</th>
<th>0.12</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time (seconds)</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Outline

• Mini-bucket elimination
• Weighted Mini-bucket
• Mini-clustering
• Iterative Belief propagation
• Iterative-join-graph propagation
• Re-parameterization, cost-shifting
Iterative Belief Propagation

- Belief propagation is exact for poly-trees
- IBP - applying BP iteratively to cyclic networks
- No guarantees for convergence
- Works well for many coding networks
- Lets combine iterative-nature with anytime--IJGP
Iterative Join Graph Propagation

- Loopy Belief Propagation
  - Cyclic graphs
  - Iterative
  - Converges fast in practice (no guarantees though)
  - Very good approximations (e.g., turbo decoding, LDPC codes, SAT – survey propagation)

- Mini-Clustering(i)
  - Tree decompositions
  - Only two sets of messages (inward, outward)
  - Anytime behavior – can improve with more time by increasing the i-bound

- We want to combine:
  - Iterative virtues of Loopy BP
  - Anytime behavior of Mini-Clustering(i)
IJGP - The basic idea

• Apply Cluster Tree Elimination to any join-graph

• We commit to graphs that are I-maps

• Avoid cycles as long as I-mapness is not violated

• Result: use minimal arc-labeled join-graphs
Tree Decomposition for Belief Updating
Tree Decomposition for belief updating
CTE: Cluster Tree Elimination

Time: $O(\exp(w+1))$
Space: $O(\exp(sep))$

For each cluster $P(X|e)$ is computed, also $P(e)$
Example

A tree decomposition for a belief network $BN = \langle X, D, G, P \rangle$ is a triple $\langle T, \chi, \psi \rangle$, where $T = (V, E)$ is a tree and $\chi$ and $\psi$ are labeling functions, associating with each vertex $v \in V$ two sets, $\chi(v) \subseteq X$ and $\psi(v) \subseteq P$ satisfying:

1. For each function $p_i \in P$ there is exactly one vertex such that $p_i \in \psi(v)$ and $\text{scope}(p_i) \subseteq \chi(v)$
2. For each variable $X_i \in X$ the set $\{v \in V | X_i \in \chi(v)\}$ forms a connected subtree (running intersection property)

Belief network

Tree decomposition

\[
\begin{align*}
\text{A} & \quad \text{B} \\
\text{B} & \quad \text{C} \quad \text{D} \\
\text{C} & \quad \text{D} \\
\text{D} & \quad \text{E} \\
\text{E} & \quad \text{F} \quad \text{G}
\end{align*}
\]
IJGP - The basic idea

- Apply Cluster Tree Elimination to any *join-graph*
- We commit to graphs that are *I-maps*
- Avoid cycles as long as I-mapness is not violated
- Result: use *minimal arc-labeled* join-graphs
Minimal Arc-Labeled Decomposition

- Use a DFS algorithm to eliminate cycles relative to each variable

a) Fragment of an arc-labeled join-graph

a) Shrinking labels to make it a minimal arc-labeled join-graph
Minimal arc-labeled join-graph

Figure 1.17: a) A belief network; b) A dual join-graph with singleton labels; c) A dual join-graph which is a join-tree

Figure 1.15: An arc-labeled decomposition
Message propagation

Minimal arc-labeled:
sep(1,2)={D,E}
elim(1,2)={A,B,C}

Non-minimal arc-labeled:
sep(1,2)={C,D,E}
elim(1,2)={A,B}

\[
\begin{align*}
    h_{(1,2)}(de) &= \sum_{a,b,c} p(a) p(c) p(b \mid ac) p(d \mid a b e) p(e \mid b c) h_{(3,1)}(bc) \\
    h_{(1,2)}(cde) &= \sum_{a,b} p(a) p(c) p(b \mid ac) p(d \mid a b e) p(e \mid b c) h_{(3,1)}(bc)
\end{align*}
\]
IJGP - Example

Belief network

Loopy BP graph
Arcs labeled with any single variable should form a TREE
Collapsing Clusters
Join-Graphs

more accuracy

less complexity
Bounded decompositions

• We want arc-labeled decompositions such that:
  • the cluster size (internal width) is bounded by \( i \) (the accuracy parameter)

• Possible approaches to build decompositions:
  • partition-based algorithms - inspired by the mini-bucket decomposition
  • grouping-based algorithms
Constructing Join-Graphs

G: (GFE)
E: (EBF) → (EF)
F: (FCD) → (BF)
D: (DB) → (CD)
C: (CAB) → (CB)
B: (BA) → (AB) → (B)
A: → (A)

a) schematic mini-bucket(i), i=3  
b) arc-labeled join-graph decomposition
IJGP properties

• IJGP(i) applies BP to min arc-labeled join-graph, whose cluster size is bounded by \( i \)

• On join-trees IJGP finds exact beliefs

• IJGP is a Generalized Belief Propagation algorithm (Yedidia, Freeman, Weiss 2001)

• Complexity of one iteration:
  • time: \( O(deg \cdot (n+N) \cdot d^{i+1}) \)
  • space: \( O(N \cdot d^0) \)
Empirical evaluation

• Algorithms:
  • Exact
  • IBP
  • MC
  • IJGP

• Measures:
  • Absolute error
  • Relative error
  • Kulbach-Leibler (KL) distance
  • Bit Error Rate
  • Time

• Networks (all variables are binary):
  • Random networks
  • Grid networks (MxM)
  • CPCS 54, 360, 422
  • Coding networks
Coding Networks – Bit Error Rate

- For N=400, 500 instances, 30 it, w*=43, σ = .22
- BER: [0.00237, 0.00238, 0.00239, 0.00240, 0.00241, 0.00242, 0.00243]
- i-bound: [0, 2, 4, 6, 8, 10, 12]

- For N=400, 500 instances, 30 it, w*=43, σ = .51
- BER: [0.0745, 0.0750, 0.0755, 0.0760, 0.0765, 0.0770, 0.0775, 0.0780, 0.0785]
- i-bound: [0, 2, 4, 6, 8, 10, 12]

- For N=400, 500 instances, 30 it, w*=43, σ = .65
- BER: [0.1900, 0.1902, 0.1904, 0.1906, 0.1908, 0.1910, 0.1912, 0.1914]
- i-bound: [0, 2, 4, 6, 8, 10, 12]

- For N=400, 1000 instances, 30 it, w*=43, σ = .22
- BER: [1e-5, 1e-4, 1e-3, 1e-2, 1e-1]
- i-bound: [0, 2, 4, 6, 8, 10, 12]

- Summary: Different values of σ show varying BER and i-bound results.
CPCS 422 – KL Distance

CPCS 422, evid=0, w*=23, 1instance

KL distance

IJGP 30 it (at convergence)
MC
IBP 10 it (at convergence)

CPCS 422, evid=30, w*=23, 1instance

KL distance

IJGP at convergence
MC
IBP at convergence

evidence=0
evidence=30
CPCS 422 – KL vs. Iterations

CPCS 422, evid=0, w*=23, 1instance

CPCS 422, evid=30, w*=23, 1instance

evidence=0

evidence=30
Coding networks - Time

Coding, N=400, 500 instances, 30 iterations, w*=43

Time (seconds)

IJGP 30 iterations
MC
IBP 30 iterations

i-bound
More On the Power of Belief Propagation

• BP as local minima of KL distance (Read Darwiche)
• BP’s power from constraint propagation perspective.
The Kullback-Leibler Divergence

The Kullback-Leibler divergence (KL–divergence)

\[ KL(Pr'(X|e), Pr(X|e)) = \sum_x Pr'(x|e) \log \frac{Pr'(x|e)}{Pr(x|e)} \]

- KL(Pr'(X|e), Pr(X|e)) is non-negative
- equal to zero if and only if Pr'(X|e) and Pr(X|e) are equivalent.
The Kullback-Leibler Divergence

KL–divergence is not a true distance measure in that it is not symmetric. In general:

$$\text{KL}(\Pr'(X|e), \Pr(X|e)) \neq \text{KL}(\Pr(X|e), \Pr'(X|e)).$$

- $\text{KL}(\Pr'(X|e), \Pr(X|e))$ weighting the KL–divergence by the approximate distribution $\Pr'$
- We shall indeed focus on the KL–divergence weighted by the approximate distribution as it has some useful computational properties.
The Kullback-Leibler Divergence

Let $\Pr(X)$ be a distribution induced by a Bayesian network $\mathcal{N}$ having families $X\cup\mathcal{U}$.

The KL–divergence between $\Pr$ and another distribution $\Pr^\prime$ can be written as a sum of three components:

$$
\text{KL}(\Pr^\prime(X|e), \Pr(X|e)) = -\text{ENT}^\prime(X|e) - \sum_{X\cup\mathcal{U}} \text{AVG}^\prime(\log \lambda_e(X)\Theta_{X|\mathcal{U}}) + \log \Pr(e),
$$

where

- $\text{ENT}^\prime(X|e) = -\sum_X \Pr^\prime(x|e) \log \Pr^\prime(x|e)$ is the entropy of the conditioned approximate distribution $\Pr^\prime(X|e)$.

- $\text{AVG}^\prime(\log \lambda_e(X)\Theta_{X|\mathcal{U}}) = \sum_{X\cup\mathcal{U}} \Pr^\prime(x|u|e) \log \lambda_e(x)\theta_{x|u}$ is a set of expectations over the original network parameters weighted by the conditioned approximate distribution.
A distribution \( \Pr'(X|e) \) minimizes the KL-divergence
\( \text{KL}(\Pr'(X|e), \Pr(X|e)) \) if it maximizes

\[
\text{ENT}'(X|e) + \sum_{X \in U} \text{AVG}'(\log \lambda_e(X) \Theta_{X|U})
\]

Competing properties of \( \Pr'(X|e) \) that minimize the KL–divergence:

- \( \Pr'(X|e) \) should match the original distribution by giving more weight to more likely parameters \( \lambda_e(x) \theta_{x|u} \) (i.e., maximize the expectations).

- \( \Pr'(X|e) \) should not favor unnecessarily one network instantiation over another by being evenly distributed (i.e., maximize the entropy).
Theorem: Yedidia, Frieman and Weiss 2005

Let $\Pr(\mathbf{X})$ be a distribution induced by a Bayesian network $\mathcal{N}$ having families $\mathbf{XU}$. Then IBP messages are a fixed point if and only if IBP marginals $\mu_u = \text{BEL}(u)$ and $\mu_{xu} = \text{BEL}(xu)$ are a stationary point of:

$$\text{ENT}'(\mathbf{X}|e) + \sum_{\mathbf{XU}} \text{AVG}'(\log \lambda_e(\mathbf{X}) \Theta_{\mathbf{XU}})$$

$$= - \sum_{\mathbf{XU}} \sum_{xu} \mu_{xu} \log \frac{\mu_{xu}}{\prod_{u \sim u} \mu_u} + \sum_{\mathbf{XU}} \sum_{xu} \mu_{xu} \log \lambda_e(x) \theta_{x|u},$$

under normalization constraints:

$$\sum_u \mu_u = \sum_{xu} \mu_{xu} = 1$$

for each family $\mathbf{XU}$ and parent $U$, and under consistency constraints:

$$\sum_{xu \sim y} \mu_{xu} = \mu_y$$

for each family instantiation $xu$ and value $y$ of family member $Y \in \mathbf{XU}$. 
Optimizing the KL-Divergence

- IBP fixed points are stationary points of the KL–divergence: they may only be local minima, or they may not be minima.
- When IBP performs well, it will often have fixed points that are indeed minima of the KL–divergence.
- For problems where IBP does not behave as well, we will next seek approximations $P_\gamma$ whose factorizations are more expressive than that of the polytree-based factorization.
Iterative Joingraph Propagation

Let $\Pr(X)$ be a distribution induced by a Bayesian network $\mathcal{N}$ having families $\mathcal{X}\mathcal{U}$, and let $C_i$ and $S_{ij}$ be the clusters and separators of a joingraph for $\mathcal{N}$.

Then messages $M_{ij}$ are a fixed point of IJGP if and only if IJGP marginals $\mu_{c_i} = BEL(c_i)$ and $\mu_{s_{ij}} = BEL(s_{ij})$ are a stationary point of:

\[
\text{ENT}'(X|\alpha) + \sum_{C_i} \text{AVC}'(\log \Phi_i)
= - \sum_{C_i} \sum_{c_i} \mu_{c_i} \log \mu_{c_i} + \sum_{S_{ij}} \sum_{s_{ij}} \mu_{s_{ij}} \log \mu_{s_{ij}} + \sum_{C_i} \sum_{c_i} \mu_{c_i} \log \Phi_i(c_i),
\]

under normalization constraints:

\[
\sum_{c_i} \mu_{c_i} = \sum_{s_{ij}} \mu_{s_{ij}} = 1
\]

for each cluster $C_i$ and separator $S_{ij}$, and under consistency constraints:

\[
\sum_{c_i \sim s_{ij}} \mu_{c_i} = \mu_{s_{ij}} = \sum_{c_j \sim s_{ij}} \mu_{c_j}
\]

for each separator $S_{ij}$ and neighboring clusters $C_i$ and $C_j$. 
A spectrum of approximations.

IBP: results from applying IJGP to the dual joingraph.

Jointree algorithm: results from applying IJGP to a jointree (as a joingraph).

In between these two ends, we have a spectrum of joingraphs and corresponding factorizations, where IJGP seeks stationary points of the KL–divergence between these factorizations and the original distribution.
Outline

• Mini-bucket elimination
• Weighted Mini-bucket
• Mini-clustering
• Iterative Belief propagation
• Iterative-join-graph propagation
• Re-parameterization, cost-shifting
Cost-Shifting

(Reparameterization)

Modify the individual functions

- but –

keep the sum of functions the same
**Tightening the bound**

- Reparameterization (or, “cost shifting”)
  - Decrease bound without changing overall function

\[
\begin{align*}
\text{max}_{a,b} f_1(a, b) + \lambda_{B\to AB}(b) + \lambda_{B\to BC}(b) &= f_{AB}(a, b) + f_{BC}(b, c) \\
\lambda_{B\to AB}(b) + \lambda_{B\to BC}(b) &= 0 \\
\text{(Adjusting functions cancel each other)}
\end{align*}
\]

(Decomposition bound is exact)
Dual Decomposition

\[ F^* = \min_x \sum_\alpha f_\alpha(x) \geq \sum_\alpha \min_x f_\alpha(x) \]

- Bound solution using decomposed optimization
- Solve independently: optimistic bound
Dual Decomposition

\[ F^* = \min_x \sum_\alpha f_\alpha(x) \geq \max_{\lambda_i \to \alpha} \sum_\alpha \min_x \left[ f_\alpha(x) + \sum_{i \in \alpha} \lambda_i \to \alpha(x_i) \right] \]

- Bound solution using decomposed optimization
- Solve independently: optimistic bound
- Tighten the bound by reparameterization
  - Enforce lost equality constraints via Lagrange multipliers

Reparameterization:
\[
\forall j : \sum_{\alpha \ni j} \lambda_j \to \alpha(x_j) = 0
\]
Dual Decomposition

\[ F^* = \min_x \sum_{\alpha} f_\alpha(x) \quad \geq \quad \max_{\lambda_i \rightarrow \alpha} \left( \sum_{\alpha} \min_x \left[ f_\alpha(x) + \sum_{i \in \alpha} \lambda_{i \rightarrow \alpha}(x_i) \right] \right) \]

Many names for the same class of bounds:
- Dual decomposition [Komodakis et al. 2007]
- TRW, MPLP [Wainwright et al. 2005; Globerson & Jaakkola, 2007]
- Soft arc consistency [Cooper & Schiex, 2004]
- Max-sum diffusion [Warner 2007]
Dual Decomposition

\[ F^* = \min_x \sum_{\alpha} f_{\alpha}(x) \geq \max_{\lambda_{i \to \alpha}} \sum_{\alpha} \min_x \left[ f_{\alpha}(x) + \sum_{i \in \alpha} \lambda_{i \to \alpha}(x_i) \right] \]

Many ways to optimize the bound:

- Sub-gradient descent  \cite{Komodakis2007, Jojic2010}
- Coordinate descent  \cite{Warner2007, Globerson2007, Sontag2009, Ihler2012}
- Proximal optimization  \cite{Ravikumar2010}
- ADMM  \cite{Meshi2011, Martins2011, Forouzan2013}
Optimizing the bound

- Can optimize the bound in various ways:
  - (Sub-)gradient descent

\[
\max_x f_1(a, b) + \lambda_{B \rightarrow AB}(b) + \max_x f_2(b, c) + \lambda_{B \rightarrow BC}(b)
\]
Optimizing the bound

- Can optimize the bound in various ways:
  - (Sub-)gradient descent

\[
\max_x f_1(a, b) + \lambda_{B \rightarrow AB}(b) + \max_x f_2(b, c) + \lambda_{B \rightarrow BC}(b)
\]
Optimizing the bound

- Can optimize the bound in various ways:
  - (Sub-)gradient descent

\[
\begin{align*}
\text{max } f_1(a, b) + \lambda_{B \rightarrow AB}(b) \\
+ \text{max } f_2(b, c) + \lambda_{B \rightarrow BC}(b)
\end{align*}
\]
Optimizing the bound

• Can optimize the bound in various ways:
  • (Sub-)gradient descent

\[
\begin{align*}
\max_x f_1(a, b) + \lambda_{B\to AB}(b) &= \max_x f_1(a, b) + \lambda_{B\to AB}(b) \\
\max_x f_2(b, c) + \lambda_{B\to BC}(b) &= \max_x f_2(b, c) + \lambda_{B\to BC}(b)
\end{align*}
\]
Various Update Schemes

- Can use any decomposition updates
  - (message passing, subgradient, augmented, etc.)

- **FGLP**: Update the original factors

- **JGLP**: Update clique function of the join graph

- **MBE-MM**: Mini-bucket with moment matching
  - Apply cost-shifting within each bucket only
Factor graph Linear Programming

- Update the original factors (FGLP)
  - Tighten all factors over $x_i$ simultaneously
  - Compute $\text{max-marginals}$
    \[
    \forall \alpha, \quad \gamma_\alpha(x_i) = \max_{x_\alpha \setminus x_i} f_\alpha
    \]
  - & update:
    \[
    \forall \alpha, \quad f_\alpha(x_\alpha) \leftarrow f_\alpha(x_\alpha) - \gamma_\alpha(x_i) + \frac{1}{|F_i|} \sum_\beta \gamma_\beta(x_i)
    \]
Mini-Bucket as Decomposition [Ihler et al. 2012]

- Downward pass as cost shifting
- Can also do cost shifting within mini-buckets: “Join graph” message passing
- “Moment-matching” version: One message exchange within each bucket, during downward sweep
- Optimal bound defined by cliques (“regions”) and cost-shifting f’n scopes (“coordinates”)

Join graph:

A: \{A\}
B: \{A,B,C\}, \{B\}
C: \{A,C,E\}, \{A,C\}
D: \{A,D,E\}, \{A,E\}
E: \{A,E\}, \{A\}

U = upper bound
MBE-MM: MBE with moment matching

**MPE** is an upper bound on MPE --U
Generating a solution yields a lower bound--L

- **Bucket A**: \( P(A) \) \( h^A(A) \) \( h^D(A) \) \( W=2 \)
- **Bucket B**: \( P(E|B,C) \) \( m_{11} \) \( P(B|A) \) \( P(D|A,B) \)
- **Bucket C**: \( P(A) \) \( P(B|A) \) \( P(D|A,B) \)
- **Bucket D**: \( h^B(C,E) \) \( h^B(A,D) \)
- **Bucket E**: \( E = 0 \) \( h^C(A,E) \)

\( \max_B \Pi \)

\( m_{11}, m_{12} \) - moment-matching messages
Algorithm 26: Algorithm MBE-MM

Input: A graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F}, \Sigma)$, variable order $o = \{X_1, \ldots, X_n\}$, i-bound parameter $i$
Output: Upper bound on the optimum value of MPE cost

// Initialize:
1. Partition the functions in $\mathbf{F}$ into $\mathbf{B}_{X_1}, \ldots, \mathbf{B}_{X_n}$, where $\mathbf{B}_{X_n}$ contains all functions $f_j$ whose highest variable is $X_n$.
   //processing bucket $\mathbf{B}_{X_k}$
2. for $k \leftarrow n$ down to 1 do
3.   Partition functions $g$ (both original and messages generated in previous buckets) in $\mathbf{B}_{X_k}$ into the mini-buckets defined $Q_{X_k} = \{q_{k1}, \ldots, q_{kg}\}$, where each $q_{ki}$ has no more than $i+1$ variables;
4.   Find the set of variables common to all the mini-buckets of variable $X_k$:
   $S_k = \text{Scope}(q_{k1}) \cap \cdots \cap \text{Scope}(q_{kg})$;
5.   Find the function of each mini-bucket $\varphi_k^g$: $\varphi_k^g \leftarrow \prod_{q_{ki} \in S_k} g$;
6.   Find the max-marginals of each mini-bucket $\varphi_k^g$: $\gamma_k^a = \max_{\text{Scope}(q_{ki})} \varphi_k^g(F_k^a)$;
7.   Update functions of each mini-bucket $F_k^a + F_k^a - \gamma_k^a + \gamma_{k-1}^a$;
8.   Generate messages $h_{X_{k-1} \rightarrow X_k}^a = \max_{X_k} F_k^a$ and place each in the bucket of highest in the ordering on variable $X_m$ in $\text{Scope}(q_{ki})$;
9. return All the buckets and the cost bound from $B_1$;

Theorem 5.3 (Complexity of MBE-MM). Given a problem with $n$ variables having domain of size $k$ and an i-bound $i$, the worst-case time complexity of MBE-MM is $O(n \cdot Q \cdot k^{i+1})$ and its space complexity is $O(n \cdot k^i)$, where $Q$ bounds the number of functions having the same variable $X_i$ in their scopes.
Anytime Approximation

- Can tighten the bound in various ways
  - Cost-shifting (improve consistency between cliques)
  - Increase i-bound (higher order consistency)
- Simple moment-matching step improves bound significantly
Anytime Approximation

- Can tighten the bound in various ways
  - Cost-shifting (improve consistency between cliques)
  - Increase i-bound (higher order consistency)
- Simple moment-matching step improves bound significantly
Anytime Approximation

- Can tighten the bound in various ways
  - Cost-shifting (improve consistency between cliques)
  - Increase i-bound (higher order consistency)
- Simple moment-matching step improves bound significantly
Outline

• Mini-bucket elimination
• Weighted Mini-bucket
• Mini-clustering
• Iterative Belief propagation
• Iterative-join-graph propagation
• Re-parameterization, cost-shifting