Bounded inference non-iteratively; Mini-bucket elimination

COMPSCI 276, Spring 2013
Set 10: Rina Dechter

(Reading: Primary: Class Notes (10)
Secondary: , Darwiche chapters 14)
Agenda

- Mini-bucket elimination
- Mini-clustering
- Iterative Belief propagation
- Iterative-join-graph propagation
Probabilistic Inference Tasks

- Belief updating:
  \[ \text{BEL}(X_i) = P(X_i = x_i \mid \text{evidence}) \]

- Finding most probable explanation (MPE)
  \[ \bar{x}^* = \arg\max_{\bar{x}} P(\bar{x}, e) \]

- Finding maximum a-posteriori hypothesis
  \[ (a_1^*, \ldots, a_k^*) = \arg\max_{\bar{a}} \sum_{\bar{x} \in A} P(\bar{x}, e) \]
  \( A \subseteq X : \) hypothesis variables

- Finding maximum-expected-utility (MEU) decision
  \[ (d_1^*, \ldots, d_k^*) = \arg\max_{\bar{d}} \sum_{\bar{x} \in D} P(\bar{x}, e)U(\bar{x}) \]
  \( D \subseteq X : \) decision variables
  \( U(\bar{x}) : \) utility function
Queries

Probability of evidence (or partition function)

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

- **Posterior marginal (beliefs):**

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

- **Most Probable Explanation**

\[ \bar{x}^* = \arg \max_{\bar{x}} P(\bar{x}, e) \]
Finding \( MPE = \max P(\overline{x}) \)

Algorithm \textit{elim-mpe} (Dechter 1996)

\[
\sum \text{ is replaced by } \max : \\
MPE = \max_{a,e,d,c,b} P(a)P(c \mid a)P(b \mid a)P(d \mid a,b)P(e \mid b,c)
\]

\( \text{max} \prod \)

Elimination operator

bucket B: \( P(b \mid a) \) \( P(d \mid b,a) \) \( P(e \mid b,c) \)

bucket C: \( P(c \mid a) \)

bucket D: \( h^C(a, d, e) \)

bucket E: \( e=0 \) \( h^D(a, e) \)

bucket A: \( P(a) \) \( h^E(a) \)

\( MPE \)

"induced width" (max clique size)
Generating the MPE-tuple

5. \( b' = \arg \max_b P(b \mid a') \times P(d' \mid b, a') \times P(e' \mid b, c') \)

4. \( c' = \arg \max_c P(c \mid a') \times h^B(a', d', c, e') \)

3. \( d' = \arg \max_d h^C(a', d, e') \)

2. \( e' = 0 \)

1. \( a' = \arg \max_a P(a) \cdot h^E(a) \)

\( B: \) \( P(b \mid a) \ P(d \mid b, a) \ P(e \mid b, c) \)

\( C: \) \( P(c \mid a) \quad h^B(a, d, c, e) \)

\( D: \) \( h^C(a, d, e) \)

\( E: \) \( e=0 \quad h^D(a, e) \)

\( A: \) \( P(a) \quad h^E(a) \)

Return \ (a', b', c', d', e') \)
**Bucket Elimination**

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

**Elimination Order:** d, e, b, c

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

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

**Original Functions**

<table>
<thead>
<tr>
<th>Original Functions</th>
<th>Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>D: ( P(d \mid a,b) )</td>
<td>( f_D(a,b) = \sum_{d} P(d \mid a,b) )</td>
</tr>
<tr>
<td>E: ( P(e \mid b,c) )</td>
<td>( f_E(b,c) = P(e = 0 \mid b,c) )</td>
</tr>
<tr>
<td>B: ( P(b \mid a) )</td>
<td>( f_B(a,c) = \sum_{b} P(b \mid a)f_D(a,b)f_E(b,c) )</td>
</tr>
<tr>
<td>C: ( P(c \mid a) )</td>
<td>( f_C(a) = \sum_{c} P(c \mid a)f_B(a,c) )</td>
</tr>
<tr>
<td>A: ( P(a) )</td>
<td>( P(a, e = 0) = p(A)f_C(a) )</td>
</tr>
</tbody>
</table>

**Bucket Tree**

- \( f_D(a,b) \)
- \( f_E(b,c) \)
- \( f_B(a,c) \)
- \( f_C(a) \)

**Time and space \( \exp(w^*) \)**
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
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.
Mini-bucket approximation:
MPE task

Split a bucket into mini-buckets => bound complexity

\[\text{bucket } (X) = \left\{ h_1, \ldots, h_r, h_{r+1}, \ldots, h_n \right\}\]

\[h^X = \max_X \prod_{i=1}^{n} h_i\]

\[\left\{ h_1, \ldots, h_r \right\} \quad \text{and} \quad \left\{ h_{r+1}, \ldots, h_n \right\}\]

\[g^X = \left( \max_X \prod_{i=1}^{r} h_i \right) \cdot \left( \max_X \prod_{i=r+1}^{n} h_i \right)\]

\[h^X \leq g^X\]

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

We can generate a solution $s$ going forward as before $U = F(s)$
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 $N'$
Relaxed network example

B1: $P(b1|a), P(d|b1,a)$
B2: $P(e|b2,c)$
C: $P(c|a)$
D:
E: $E = e$
A: $P(a)$

(a) (b)
**MBE-mpe(i)**

- **Input:** $i$ – max number of variables allowed in a mini-bucket
- **Output:** [lower bound (Probability of a sub-optimal solution), upper bound]

**Example:** `approx-mpe(3) versus elim-mpe`

- Mini-buckets: $P(\text{elb},c)$, $P(\text{dla},b)P(\text{bla})$, $P(\text{cla})$, $h^B(e,c)$, $h^B(d,a)$, $E = 0$, $h^C(e,a)$, $P(a)$, $h^E(a)$, $h^D(a)$
- Max variables in a mini-bucket: $3$, $3$, $2$, $2$, $1$
- $w^* = 2$

- $U = \text{Upper bound (MPE)}$

- Upper bounds:
  - $P(\text{elb},c)$
  - $P(\text{dla},b)P(\text{bla})$
  - $P(\text{cla})$
  - $h^B(a,d,c,e)$
  - $h^C(a,d,e)$
  - $h^E(a,e)$
  - $h^D(a,e)$

- $MPE$
- $w^* = 4$
(i,m) partitionings

Definition 7.1.1 ((i,m)-partitioning) Let $H$ be a collection of functions $h_1, ..., h_t$ defined on scopes $S_1, ..., S_t$, 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 $h_1, ..., h_t$ 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)
- **Process each bucket from** \(p=n\) **to** 1
  - Create \((i,m)\)-mini-buckets partitions
  - Process each mini-bucket
- **(For mpe):** assign values in ordering \(d\)
- **Return:** mpe-tuple, upper and lower bounds
Algorithm mbe-mpe(i, m)
Input: A belief network $BN = (G, P)$, an ordering $o$, evidence $\tilde{e}$.
Output: An upper bound $U$ and a lower bound $L$ on the $MPE = \max_{\bar{x}} P(\bar{x}, \tilde{e})$, and a suboptimal solution $\bar{x}^a$ that provides $L = P(\bar{x}^a)$.
1. Initialize: Partition $P = \{P_1, \ldots, P_n\}$ into buckets $bucket_1, \ldots, bucket_n$, where $bucket_p$ contains all CPTs $h_1, h_2, \ldots, h_t$ whose highest-index variable is $X_p$.
2. Backward: for $p = n$ to 2 do
   - If $X_p$ is observed ($X_p = a$), assign $X_p = a$ in each $h_j$ and put the result in its highest-variable bucket (put constants in $bucket_1$).
   - Else for $h_1, h_2, \ldots, h_t$ in $bucket_p$ do
     Generate an $(i, m)$-mini-bucket-partitioning, $Q' = \{Q_1, \ldots, Q_r\}$.
     for each $Q_l \in Q'$ containing $h_{t_1}, \ldots, h_{t_k}$, do
       compute $h^l = \max_{X_p} \prod_{j=1}^{t_k} h_{ij}$ and place it in the bucket of the highest-index variable in $U_t \leftarrow \bigcup_{j=1}^{t_k} S_{ij} - \{X_p\}$, where $S_{ij}$ is the scope of $h_{ij}$ (put constants in $bucket_1$).
3. Forward: for $p = 1$ to $n$, given $x_1^a, \ldots, x_{p-1}^a$, do
   assign a value $x_p^a$ to $X_p$ that maximizes the product of all functions in $bucket_p$.
4. Return the assignment $\bar{x}^a = (x_1^a, \ldots, x_n^a)$, a lower bound $L = P(\bar{x}^a)$, and an upper bound $U = \max_{x_1} \prod_{h_{ij} \in bucket_1} h^j$ on the $MPE = \max_{\bar{x}} P(\bar{x}, \tilde{e})$.

Theorem 7.1.3 (mbe-mpe properties) Algorithm mbe-mpe(i, m) computes an upper bound on the MPE. Its time and space complexity is $O(n \cdot \exp(i))$ where $i \leq n$. 

17
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 $h^p \leq g^p_{Q'} \leq g^p_{Q''}$.

Remember that mbe-mpe computes the bounds on $MPE = \max_{\bar{x}} P(\bar{x}, \bar{e})$, rather than on $M = \max_{\bar{x}} P(\bar{x}|\bar{e}) = MPE/P(\bar{e})$. Thus

$$\frac{L}{P(\bar{e})} \leq M \leq \frac{U}{P(\bar{e})}$$
Properties of MBE-mpe(i)

- **Complexity:** $O(\exp(i))$ time and $O(\exp(i))$ space.

- **Accuracy:** determined by upper/lower (U/L) bound.

- As $i$ increases, both accuracy and complexity increase.

- Possible use of mini-bucket approximations:
  - As *anytime algorithms*
  - As *heuristics* in best-first search
Anytime Approximation

Algorithm anytime-mpe(ε)
Input: Initial values of $i$ and $m$, $i_0$ and $m_0$; increments $i_{step}$ and $m_{step}$, and desired approximation error $ε$.
Output: $U$ and $L$
1. Initialize: $i = i_0, m = m_0$.
2. do
3. run mbe-mpe($i, m$)
4. $U \leftarrow$ upper bound of mbe-mpe($i, m$)
5. $L \leftarrow$ lower bound of mbe-mpe($i, m$)
6. Retain best bounds $U, L$, and best solution found so far
7. if $1 \leq U/L \leq 1 + ε$, return solution
8. else increase $i$ and $m$: $i \leftarrow i + i_{step}$ and $m \leftarrow m + m_{step}$
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

- 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)

- MBE-bel-max(i,m), MBE-bel-min(i,m) generating upper and lower-bound on beliefs approximates BE-bel

- MBE-map(i,m): max buckets will be maximized, sum buckets will be sum-max. Approximates BE-map.
Normalization

- mbe-bel computes upper/lower bound on the joint marginal distributions.

Alternatively, let $U_i$ and $L_i$ be the upper bound and lower bounding functions on $P(X_1 = x_i, \bar{e})$ obtained by $mbe-bel-max$ and $mbe-bel-min$, respectively. Then,

$$\frac{L_i}{P(\bar{e})} \leq P(x_i|\bar{e}) \leq \frac{U_i}{P(\bar{e})}$$

We sometime use normalization of the approximation, but then no guarantee. The probable is that we have to approximate also the partition function.
Algorithm mbe-bel-max(i,m)

Input: A belief network $BN = (G, P)$, an ordering $o$, and evidence $\vec{e}$.
Output: an upper bound on $P(x_1, \vec{e})$ and an upper bound on $P(e)$.

1. Initialize: Partition $P = \{P_1, ..., P_n\}$ into buckets $bucket_1, ..., bucket_n$, where $bucket_k$ contains all CPTs $h_1, h_2, ..., h_t$ 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 $h_1, h_2, ..., h_t$ in $bucket_k$ do
     Generate an $(i, m)$-mini-bucket-partitioning, $Q' = \{Q_1, ..., Q_r\}$.
     For each $Q_l \in Q'$, containing $h_{t_1}, ..., h_{t_k}$, do
       If $l = 1$ compute $h^l = \sum_{X_k} \prod_{j=1}^{t} h_{1_j}$
       Else compute $h^l = \max_{X_k} \prod_{j=1}^{t} h_{t_j}$
       Add $h^l$ to the bucket of the highest-index variable in $U_l \leftarrow \bigcup_{j=1}^{t} S_{i_j} - \{X_k\}$, (put constant functions in $bucket_1$).

3. Return $P'(\vec{x}_1, e) < --$ the product of functions in the bucket of $X_1$, which is an upper bound on $P(x_1, \vec{e})$.
$P'(e) < -- \sum_{x_1} P'(\vec{x}_1, e)$, which is an upper bound on probability of evidence.
Empirical Evaluation
(Dechter and Rish, 1997; Rish thesis, 1999)

- Randomly generated networks
  - Uniform random probabilities
  - Random noisy-OR
- CPCS networks
- Probabilistic decoding

Comparing MBE-mpe and anytime-mpe versus BE-mpe
Methodology for Empirical Evaluation (for mpe)

- U/L – accuracy
- Better (U/mpe) or mpe/L
- Benchmarks: Random networks
  - Given n, e, v generate a random DAG
  - For xi and parents generate table from uniform [0,1], or noisy-or
- Create k instances. For each, generate random evidence, likely evidence
- Measure averages
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>
The effect of evidence

More likely evidence => higher MPE => higher accuracy (why?)

Log(U/L) histogram for i=10 on 1000 instances of likely evidence

Log(U/L) histogram for i=10 on 1000 instances of random evidence

Likely evidence versus random (unlikely) evidence
Algorithm mbe-map(i,m)

Input: A belief network BN = (G, P), a subset of variables A = \{A_1,\ldots,A_k\}, an ordering of the variables, o, in which the A’s appear first, and evidence \(\bar{e}\).

Output: An upper bound \(\bar{U}\) on the MAP and a suboptimal solution \(\bar{A} = \bar{a}_k\).

1. Initialize: Partition \(P = \{P_1,\ldots,P_n\}\) into buckets \(bucket_1,\ldots,bucket_n\) where \(bucket_p\) contains all CPTs, \(h_1,\ldots,h_t\) whose highest index variable is \(X_p\).

2. Backward: for \(p = n\) to 1 do
   - If \(X_p\) is observed \((X_p = a)\), assign \(X_p = a\) in each \(h_i\) and put the result in its highest-variable bucket (put constants in \(bucket_1\)).
   - Else for \(h_1, h_2,\ldots,h_t\) in \(bucket_p\) do
     - Generate an \((i, m)\)-partitioning, \(Q’\) of the matrixes \(h_i\) into mini-buckets \(Q_1,\ldots,Q_r\).
     - If \(X_P \notin A\) /* not a hypothesis variable */ for each \(Q_l \in Q’\), containing \(h_{t_l},\ldots,h_{t_r}\), do
       - If \(l = 1\), compute \(h^l = \sum_{X_P} \Pi_{i=1}^{t_l} h_{i}\)
       - Else compute \(h^l = \max_{X_P} \Pi_{i=1}^{t_l} h_{i}\)
       - Add \(h^l\) to the bucket of the highest-index variable in \(U_l \leftarrow \bigcup_{i=1}^{t_l} S_i - \{X_P\}\) (put constants in \(bucket_1\)).
     - Else \((X_P \in A)\) /* a hypothesis variable */ for each \(Q_l \in Q’\) containing \(h_{t_l},\ldots,h_{t_r}\) compute \(h^l = \max_{X_P} \Pi_{i=1}^{t_l} h_{i}\) and place it in the bucket of the highest-index variable in \(U_l \leftarrow \bigcup_{i=1}^{t_l} S_i - \{X_P\}\) (put constants in \(bucket_1\)).
   - Else \((X_P \in A)\) /* a hypothesis variable */ for each \(Q_l \in Q’\) containing \(h_{t_l},\ldots,h_{t_r}\) compute \(h^l = \max_{X_P} \Pi_{i=1}^{t_l} h_{i}\) and place it in the bucket of the highest-index variable in \(U_l \leftarrow \bigcup_{i=1}^{t_l} S_i - \{X_P\}\) (put constants in \(bucket_1\)).

3. Forward: for \(p = 1\) to \(k\), given \(A_1 = a_1^e,\ldots,A_{p-1} = a_{p-1}^e\), assign a value \(a_p^e\) to \(A_p\) that maximizes the product of all functions in \(bucket_p\).

4. Return An upper bound \(\bar{U} = \max_{a_1} \Pi_{h \in bucket_1} h_i\) on MAP, computed in the first bucket.

and the assignment \(\bar{a}_k = (a_1^e,\ldots,a_k^e)\).

Figure 7.6: Algorithm mbe-map(i,m).
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)
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 $X_j$ 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 $Y = (Y^u, Y^x)$. The decoding task is to assess the most likely values for the $U$'s given the observed values $Y = (\tilde{y}^u, \tilde{y}^x)$, which is the MAP task where $U$ is the set of hypothesis variables, and $Y = (\tilde{y}^u, \tilde{y}^x)$ is the evidence. After processing the observed buckets we get the following bucket configuration (lower case $y$'s are observed values):

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

Processing by $\text{mb-}\text{-map}(4, 1)$ of the first top five buckets by summation and the rest by maximization, results in the following mini-bucket partitionings and function generation:
\[ \text{bucket}(X_0) = \{ P(y_X^0 | X_0), P(X_0 | U_0, U_1, U_2) \}, \]
\[ \text{bucket}(X_1) = \{ P(y_X^1 | X_1), P(X_1 | U_1, U_2, U_3) \}, \]
\[ \text{bucket}(X_2) = \{ P(y_X^2 | X_2), P(X_2 | U_2, U_3, U_4) \}, \]
\[ \text{bucket}(X_3) = \{ P(y_X^3 | X_3), P(X_3 | U_3, U_4, U_0) \}, \]
\[ \text{bucket}(X_4) = \{ P(y_X^4 | X_4), P(X_4 | U_4, U_0, U_1) \}, \]
\[ \text{bucket}(U_0) = \{ P(U_0), P(y_X^0 | U_0), h^{X_0}(U_0, U_1, U_2) \}, \{ h^{X_2}(U_3, U_4, U_0) \}, \{ h^{X_4}(U_4, U_0, U_1) \}, \]
\[ \text{bucket}(U_1) = \{ P(U_1), P(y_X^1 | U_1), h^{X_1}(U_1, U_2, U_3), h^{U_0}(U_1, U_2) \}, \{ h^{U_4}(U_4, U_1) \}, \]
\[ \text{bucket}(U_2) = \{ P(U_2), P(y_X^2 | U_2), h^{X_2}(U_2, U_3, U_4), h^{U_1}(U_2, U_3) \}, \]
\[ \text{bucket}(U_3) = \{ P(U_3), P(y_X^3 | U_3), h^{U_0}(U_3, U_4), h^{U_1}(U_3, U_4), h^{U_2}(U_3, U_4) \}, \]
\[ \text{bucket}(U_4) = \{ P(U_4), P(y_X^4 | U_4), h^{U_4}(U_4) \}, \{ h^{U_2}(U_4) \}. \]

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 \( U_3 \). 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 \( U \) satisfying \( U \geq MAP = P(U, \bar{y}^u, \bar{y}^v) \) where \( \bar{y}^u \) and \( \bar{y}^v \) are the observed outputs for the \( U \)'s and the \( X \)'s bits transmitted. In order to bound \( P(U | \bar{e}) \), where \( \bar{e} = (\bar{y}^u, \bar{y}^v) \), 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 \( P(U_1, \bar{e}) / P(U_2, \bar{e}) \). \( \square \)
Complexity and tractability of MBE(i,m)

**Theorem 7.6.1** Algorithm mbe(i,m) takes $O(r \cdot \exp(i))$ time and space, where $r$ is the number of input functions, and where $|F|$ is the maximum scope of any input function, $|F| \leq i \leq n$. For $m = 1$, the algorithm is time and space $O(r \cdot \exp(|F|))$. 
Belief propagation is easy on polytree: Pearl’s Belief Propagation

A polytree: a tree with larger families

A polytree decomposition

- Running CTE = running Pearl’s BP over the dual graph
- Dual-graph: nodes are cpts, arcs connect non-empty intersections. BP is Time and space linear
Iterative Belief Propagation

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

One step:
CTE - bel(U₁)

- No guarantees for convergence
- Works well for many coding networks
MBE-mpe vs. IBP

MBE-mpe is better on low - w* codes
IBP 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
Mini-buckets: summary

- Mini-buckets – local inference approximation

- Idea: bound size of recorded functions

- MBE-mpe(i) - mini-bucket algorithm for MPE
  - Better results for noisy-OR than for random problems
  - Accuracy increases with decreasing noise in coding
  - Accuracy increases for likely evidence
  - Sparser graphs -> higher accuracy
  - Coding networks: MBE-mpe outperforms IBP on low-induced width codes
Agenda

- Mini-bucket elimination
- Mini-clustering
- Iterative Belief propagation
- Iterative-join-graph propagation
Cluster Tree Elimination - properties

- Correctness and completeness: Algorithm CTE is correct, i.e. it computes the exact joint probability of a single variable and the evidence.

- Time complexity: $O(\deg \times (n+N) \times d^{w*+1})$

- Space complexity: $O(N \times d^{sep})$

where
- $\deg$ = the maximum degree of a node
- $n$ = number of variables (= number of CPTs)
- $N$ = number of nodes in the tree decomposition
- $d$ = the maximum domain size of a variable
- $w^*$ = the induced width
- $sep$ = the separator size
Join-Tree Clustering

**EXACT algorithm**

**Time and space:**
\[ \exp(\text{cluster size}) = \exp(\text{treewidth}) \]

1. \[ h_{(1,2)}(b, c) = \sum_a p(a) \cdot p(b \mid a) \cdot p(c \mid a, b) \]
2. \[ h_{(2,1)}(b, c) = \sum_{d, f} p(d \mid b) \cdot p(f \mid c, d) \cdot h_{(3,2)}(b, f) \]
3. \[ h_{(3,2)}(b, f) = \sum_e p(e \mid b, f) \cdot h_{(4,3)}(e, f) \]
4. \[ h_{(4,3)}(e, f) = p(G = g_e \mid e, f) \]
Mini-Clustering

Split a cluster into mini-clusters => bound complexity

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

Exponential complexity decrease \[ O(e^n) \rightarrow O(e^{\text{var}(r)}) + O(e^{\text{var}(n-r)}) \]
Mini-Clustering, i-bound=3

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

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

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

**Approximate algorithm**

Time and space:

\[ \exp(i\text{-bound}) \]

Number of variables in a mini-cluster
Semantic of variable duplication for mini-clustering

We can have a different duplication of nodes going up and down. Example: going down (left) and up (right)

Figure 1.14: Node duplication semantics of MC: (a) trace of MC-BU(3); (b) trace of CTE-BU.
Mini-Clustering

- **Correctness and completeness**: Algorithm MC-bel(i) computes a bound (or an approximation) on the joint probability \( P(X_i, e) \) of each variable and each of its values.

- **Time & space complexity**: \( O(n \times hw^* \times k_i) \)

where \( hw^* = \max_u \{ f \mid f \cap \chi(u) \neq \phi \} \)
We can replace $\max$ operator by

- $\min$  $\Rightarrow$  lower bound on the joint
- $\text{mean}$  $\Rightarrow$  approximation of the joint
Grid 15x15 - 10 evidence

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

- NHD
- Absolute error
- Relative error
- Time (seconds)
CPCS422 - Absolute error

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

CPCS 422, evid=10, w*=23, 1 instance

evidence=0

evidence=10
Coding networks - Bit Error Rate

Coding networks, $N=100$, $P=4$, $\sigma=.22$, $w^*=12$, 50 instances

Coding networks, $N=100$, $P=4$, $\sigma=.51$, $w^*=12$, 50 instances

$\sigma=0.22$

$\sigma=.51$
**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 minibucket 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. Then, each minibucket is absorbed into the left-most mini-bucket with whom it can be merged.

The time and space complexity of $\text{Partition}(B, i)$, where $B$ is the partitioned bucket, using the SCP heuristic is $O(|B| \log |B|) + |B|^2$ and $O(\exp(i))$, respectively.

The scope-based heuristic is quite fast, its shortcoming is that it does not consider the actual information in the functions.
Content-based heuristics
(Rollon and Dechter 2010)

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

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

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

Use greedy heuristic derived from a distance function to decide which functions go into a single mini-bucket
Agenda

- Mini-bucket elimination
- Mini-clustering
- Iterative Belief propagation
- Iterative-join-graph propagation
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 \textit{join-graph}
- We commit to graphs that are \textit{I-maps}
- Avoid cycles as long as I-mapness is not violated
- Result: use \textit{minimal arc-labeled} join-graphs
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
IJGP - Example

Belief network

Loopy BP graph
Arcs labeled with any single variable should form a TREE
Collapsing Clusters
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 | ac) p(d | abe) p(e | bc) h_{(3,1)}(bc) \\
    h_{(1,2)}(cde) &= \sum_{a,b} p(a) p(c) p(b | ac) p(d | abe) p(e | bc) h_{(3,1)}(bc)
\end{align*}
Bounded decompositions

- We want arc-labeled decompositions such that:
  - the cluster size (internal width) is bounded by $i$ (the accuracy parameter)
  - the width of the decomposition as a graph (external width) is as small as possible

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

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: $O(N \cdot d^i)$
Empirical evaluation

- Algorithms:
  - Exact
  - IBP
  - MC
  - IJGP

- Networks (all variables are binary):
  - Random networks
  - Grid networks (MxM)
  - CPCS 54, 360, 422
  - Coding networks

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

Coding, N=400, 1000 instances, 30 it, w*=43, sigma=.22

Coding, N=400, 500 instances, 30 it, w*=43, sigma=.32

Coding, N=400, 500 instances, 30 it, w*=43, sigma=.51

Coding, N=400, 500 instances, 30 it, w*=43, sigma=.65
CPCS 422 – KL vs. Iterations

In the diagrams, the KL distance is plotted against the number of iterations for two different evid values: evidence = 0 and evidence = 30. The x-axis represents the number of iterations ranging from 0 to 35, while the y-axis represents the KL distance ranging from 0.0001 to 0.1. The plots show three different methods: IJGP(3), IJGP(10), and IBP. The results indicate a decrease in KL distance as the number of iterations increases, with a more pronounced effect for evidence = 30 compared to evidence = 0.
Coding networks - Time

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

Time (seconds)

i-bound

IJGP 30 iterations
MC
IBP 30 iterations
More On the Power of Belief Propagation

- BP as local minima of KL distance
- BP’s power from constraint propagation perspective.
More On the Power of Belief Propagation

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

The Kullback-Leibler divergence (KL–divergence)

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

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

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

\[ KL(Pr'(X|e), Pr(X|e)) \neq KL(Pr(X|e), Pr'(X|e)). \]

- 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 $\Pr(\mathbf{X})$ be a distribution induced by a Bayesian network $\mathcal{N}$ having families $XU$

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

$$
\text{KL}(\Pr'(\mathbf{X}|e), \Pr(\mathbf{X}|e)) = -\text{ENT}'(\mathbf{X}|e) - \sum_{XU} \text{AVG}'(\log \lambda_e(X)\Theta_{X|U}) + \log \Pr'(e),
$$

where

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

- $\text{AVG}'(\log \lambda_e(X)\Theta_{X|U}) = \sum_{XU} \Pr'(xu|e) \log \lambda_e(x)\theta_{x|u}$ is a set of expectations over the original network parameters weighted by the conditioned approximate distribution.
The Kullback-Leibler Divergence

A distribution $\Pr'(X|e)$ minimizes the KL-divergence $KL(\Pr'(X|e), \Pr(X|e))$ 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).
Optimizing the KL-Divergence

The approximations computed by IBP are based on assuming an approximate distribution $\Pr'(\mathbf{X})$ that factors as follows:

$$
\Pr'(\mathbf{X}|\mathbf{e}) = \prod_{\mathcal{X}\cup} \frac{\Pr'(\mathbf{X}\cup|\mathbf{e})}{\prod_{\mathcal{U}\in\mathcal{U}} \Pr'(\mathbf{U}|\mathbf{e})}
$$

- This choice of $\Pr'(\mathbf{X}|\mathbf{e})$ is expressive enough to describe distributions $\Pr(\mathbf{X}|\mathbf{e})$ induced by polytree networks $\mathcal{N}$.
- In the case where $\mathcal{N}$ is not a polytree, then we are simply trying to fit $\Pr(\mathbf{X}|\mathbf{e})$ into an approximation $\Pr'(\mathbf{X}|\mathbf{e})$ as if it were generated by a polytree network.
- The entropy of distribution $\Pr'(\mathbf{X}|\mathbf{e})$ can be expressed as:

$$
\text{ENT}'(\mathbf{X}|\mathbf{e}) = - \sum_{\mathcal{X}\cup} \sum_{\mathbf{X}\cup} \Pr'(\mathbf{X}\cup|\mathbf{e}) \log \frac{\Pr'(\mathbf{X}\cup|\mathbf{e})}{\prod_{\mathcal{U}\sim \mathcal{U}} \Pr'(\mathcal{U}|\mathbf{e})}
$$
Optimizing the KL-Divergence

Let $P_t(\mathbf{X})$ 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 $\mu_u = BEL(u)$ and $\mu_{\mathbf{XU}} = BEL(\mathbf{XU})$ are a stationary point of:

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

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

under normalization constraints:

$$\sum_u \mu_u = \sum_{\mathbf{XU}} \mu_{\mathbf{XU}} = 1$$

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

$$\sum_{\mathbf{XU} \sim y} \mu_{\mathbf{XU}} = \mu_y$$

for each family instantiation $\mathbf{XU}$ 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 $P_{r'}$ whose factorizations are more expressive than that of the polytree-based factorization.
If a distribution \( P_r' \) has the form:

\[
P_r'(X|e) = \frac{\prod_c P_r'(C|e)}{\prod_s P_r'(S|e)},
\]

then its entropy has the form:

\[
E_n'(X|e) = \sum_c E_n'(C|e) - \sum_s E_n'(S|e).
\]

When the marginals \( P_r'(C|e) \) and \( P_r'(S|e) \) are readily available, the \( E_n \) component of the KL–divergence can be computed efficiently.
While a jointree induces an exact factorization of a distribution, a joingraph $G$ induces an approximate factorization:

$$
Pr'(X|e) = \frac{\prod_i Pr'(C_i|e)}{\prod_{ij} Pr'(S_{ij}|e)}
$$

which is a product of cluster marginals over a product of separator marginals. When the joingraph corresponds to a jointree, the above factorization will be exact.
A dual joingraph leads to the factorization used by IBP.
The jointree induces the following factorization, which is exact:

\[
\Pr'(X|e) = \frac{\Pr'(ABC|e)\Pr'(ABD|e)\Pr'(ABCD|e)\Pr'(CDE|e)}{\Pr'(ABC|e)\Pr'(ABD|e)\Pr'(CD|e)}
\]
The joingraph induces the following factorization:

\[
\Pr'(X|e) = \frac{\Pr'(ABC|e)\Pr'(ABD|e)\Pr'(ACD|e)\Pr'(CDE|e)}{\Pr'(B|e)\Pr'(AC|e)\Pr'(AD|e)\Pr'(CD|e)}
\]
Iterative Joinigraph Propagation

Computing cluster marginals $\mu_{c_i} = \Pr'(c_i|e)$ and separator marginals $\mu_{s_{ij}} = \Pr'(s_{ij}|e)$ that minimize the KL–divergence between $\Pr'(X|e)$ and $\Pr(X|e)$

This optimization problem can be solved using a generalization of IBP, called iterative joinigraph propagation (IJGP), which is a message passing algorithm that operates on a joinigraph.
Iterative Joingraph Propagation

\textbf{IJGP}(G, \Phi)

\textbf{input:}

- \(G\): a joingraph
- \(\Phi\): factors assigned to clusters of \(G\)

\textbf{output:} approximate marginal \(BEL(C_i)\) for each node \(i\) in the joingraph \(G\).

\textbf{main:}

1: \(t \leftarrow 0\)
2: initialize all messages \(M_{ij}^t\) (uniformly)
3: while messages have not converged do
4: \(t \leftarrow t + 1\)
5: for each joingraph edge \(i \rightarrow j\) do
6: \(M_{ij}^t \leftarrow \eta \\sum_{c_i \in S_{ij}} \Phi_i \prod_{k \neq j} M_{ki}^{t-1}\)
7: \(M_{ji}^t \leftarrow \eta \\sum_{c_j \in S_{ij}} \Phi_j \prod_{k \neq i} M_{kj}^{t-1}\)
8: end for
9: end while
10: return \(BEL(C_i) \leftarrow \eta \Phi_i \prod_k M_{ki}^t\) for each node \(i\)
Iterative Join Graph Propagation

Let \( P_{\Theta}(X) \) be a distribution induced by a Bayesian network \( \mathcal{N} \) having families \( XU \), and let \( C_i \) and \( S_{ij} \) be the clusters and separators of a join graph 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:

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

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:

\[
\sum_{C_i \sim S_{ij}} \mu_{c_i} = \sum_{C_j \sim S_{ij}} \mu_{c_j}
\]

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.