Chapter 8

Bounding Inference: mini-bucket schemes

Up to now we focused on exact algorithms for processing graphical models emphasizing the two reasoning styles of inference and search. We also showed that hybrids of AND/OR search and inference are effective and can be used to trade space for time.

Clearly, due to the hardness of the tasks, some networks cannot be processed exactly when their structure is not sparse thus having high treewidth and when their functions do not posses any internal structure. In such cases approximation algorithms are the only choice. Approximation algorithms can be designed to approximate either an inference, message-passing scheme, or a search scheme or their hybrid. Bounded inference algorithms, on which this chapter focuses, approximate inference schemes, while sampling schemes can be viewed as approximating search.

This chapter presents a class of approximation algorithms that bound the dimensionality of the dependencies created by inference. This yields a collection of parameterized schemes such as mini-buckets, mini-clustering and iterative join-graph propagation that offers adjustable trade-off between accuracy and efficiency.

It was shown that approximation scheme within a given relative error bounds, is NP-hard [77, 87]. Nevertheless there are approximation strategies that work well in practice. One alternative for dealing with these bleak fact is to develop *anytime algorithms*. Such algorithms produce better and better solutions with time and can therefore be interrupted

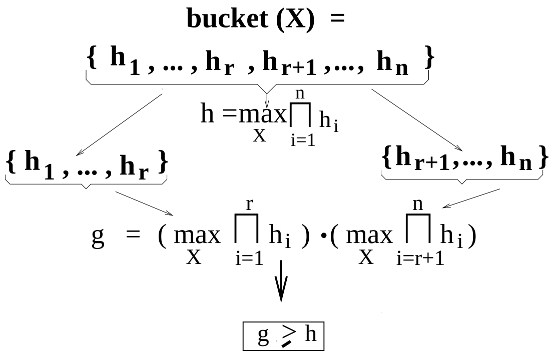

bucket (X) =
{ h1 , ... , hr , hr+1 ,..., hn }
h = max_X ∏_{i=1}^{n} hi
{ h1 , ... , hr }
{hr+1,..., hn}
g = (max_X ∏_{i=1}^{r} hi) •(max_X ∏_{i=r+1}^{n} hi)
g ≥ h

Figure 8.1: The idea of mini-bucket approximation.

at any time producing the best solution found thus far. More significantly, algorithms that provide bounding guarantees (upper and lower bounds) on the exact answer, that can be tightened with more time, are most desirable. Such algorithms are presented in this chapter.

As we showed (Chapter 4) the bucket-elimination algorithm is a unifying algorithmic scheme for variable-elimination algorithms that are widely applicable. Among the algorithms that can be expressed as bucket-elimination are *directional-resolution* for propositional satisfiability *adaptive-consistency* for constraint satisfaction, *Fourier* and *Gaussian elimination* for linear inequalities, *dynamic-programming* for combinatorial optimization as well as many algorithms for probabilistic inference [27]. In the following sections we will introduce the mini-bucket elimination scheme that approximates bucket-elimination, and is therefore applicable across all graphical models. Subsequently we will generalize the mini-bucket scheme into a scheme called weighted mini-bucket using the power-sum operator, whose weight facilitate far more extensive bound tightening. Finally, we show how augmenting weighted mini-bucket by cost-shifting (e.g. re-parameterizations) yield a rich framework for bounding graphical models tasks that lend itself to anytime schemes.

8.1 Mini-bucket approximation for MPE

Consider the bucket-elimination algorithm *BE-mpe* (Chapter 4). Since the complexity of processing a bucket depends on the number of arguments in the scope of the functions being recorded, we should consider approximating these functions by a collection of smaller-arity functions. Let $h_1, ..., h_t$ be the functions in the bucket of X_p, and let $S_1, ..., S_t$ be their scopes. When *BE-mpe* processes bucket(X_p), the function $h_p = max_{X_p} \psi_p \Pi_{i=1}^t h_i$ is computed. A simple approximation idea is to compute an upper bound on h_p by "migrating" the maximization inside the multiplication. Since, in general, for any two non-negative functions $Z(x)$ and $Y(x)$, $\max_x Z(x) \cdot Y(x) \leq \max_x Z(x) \cdot \max_x Y(x)$, this approximation will compute an upper bound on h_p. For example, the function $g_p = \max_{X_p} \psi_P \prod_{i=1}^t \max_{X_p} h_i$, is an upper bound on $h_p = max_{X_p} \psi_p \prod_{i=1}^t h_i$. Procedurally it implies that the elimination operation of maximization is applied separately to each function, requiring less computation. This is because we do not multiply functions before applying the elimination operation of maximization and thus avoid the combinatorial explosion associated with generating a function of the scope of all the bucket's variables.

The idea is demonstrated in Figure 8.1, where the bucket of variable X having n functions is split into two mini-buckets, one having r functions and the other having $n-r$ functions, where $r \leq n$. In general, any partitioning of a set of functions $h_1, ..., h_t$ can be partitioned into subsets called *mini-buckets*. Let $Q = \{Q_1, ..., Q_r\}$ be a partitioning into mini-buckets of the functions $h_1, ..., h_t$ in X_p's bucket (these are either original functions or messages sent from other buckets). Assume that the mini-bucket Q_l contains the functions $h_{l_1}, ..., h_{l_r}$. The exact *BE-mpe* algorithm computes $h_p = \max_{X_p} \prod_{i=1}^t h_i$, which can be rewritten as $h_p = \max_{X_p} \prod_{l=1}^r \prod_{h \in Q_l} h$. By migrating maximization into each mini-bucket we can compute, ionstead: $g_p = \prod_{l=1}^r \max_{X_p} \prod_{h \in Q_l} h$. Each new mini-bucket function (or message), $\max_{X_p} \prod_{h \in Q_l} h$, can now be computed independently and be placed separately into the bucket of the highest-variable in its scope. The algorithm then proceeds with the next variable. Functions without arguments (i.e., constants) are placed in the lowest bucket.

Since we replace functions with their upper bounds, it is easy to see that once all buckets are processed, the maximized product generated in the first bucket is an upper bound on the MPE value. A lower bound can also be computed as the probability of any (suboptimal) assignment. In particular the suboptimal configuration that can be generated in a forward phase, similar to the way a solution is generated by the exact algorithm, can yield a good candidate solution and its associated lower-bound.

It is convenient to control the algorithm's performance using two bounding parameters. Parameter i will bound the number of variables in each mini-bucket, while m will bound the number of its functions. The mini-bucket elimination (*MBE*) algorithm for finding an *mpe*, *MBE-mpe(i,m)*, is described in Figure 8.2.

Clearly we would want the mini-buckets to be as small as possible yet we wish the scheme to be as accurate as possible. We would want to have a small number of mini-buckets as which also means that we should assign many functions as possible to each mini-bucket. In particular, if a single function's scope subsumed another, they should share the same mini-bucket.

In general, as m and i increase, we get more accurate approximations. Note however, that a monotonic increase in accuracy as a function of i can be guaranteed only for restricted cases, as the refinement property that we discuss next, suggests.

Definition 8.1.1 (refinement) *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 8.1.2 *If $Q^{''}$ is a refinement of $Q^{'}$ in bucket$_p$, then $h^p \leq g^p_{Q'} \leq g^p_{Q''}$.*

Proof: Clearly for any partitioning Q we have $h^p \leq g^p_Q$. By definition, given a refinement $Q'' = \{Q''_1, ..., Q''_k\}$ of a partitioning $Q' = \{Q'_1, ..., Q'_m\}$, each mini-bucket $i \in \{1, ..., k\}$ of Q'' belongs to some mini-bucket $j \in \{1, ..., m\}$ of Q'. In other words, each mini-bucket j of Q' is further partitioned into the corresponding mini-buckets of Q'', $Q'_j = \{Q''_{j1}, ..., Q''_{jl}\}$. Therefore,

$$g^p_{Q''} = \prod_{i=1}^{k} (\max_{X_p} \prod_{l \in Q''_i} h_l) = \prod_{j=1}^{m} \prod_{Q''_i \subseteq Q'_j} (\max_{X_p} \prod_{l \in Q''_i} h_l) \geq \prod_{j=1}^{m} (\max_{X_p} \prod_{l \in Q'_j} h_l) = g^p_{Q'}.$$

■

```
**Algorithm MBE-mpe(i,m)**
**Input:** A belief network $\mathcal{B} =< X, D, G, \mathcal{P} >$, where $\mathcal{P} = \{P_1, ..., P_n\}$; an ordering of the variables, $d = X_1, ..., X_n$; observations $\mathbf{e}$.
**Output:** An upper bound $U$ and a lower bound $L$ on the most probable configuration given the evidence. A suboptimal solution $\bar{x}^a$ that provides the lower bound $L = P(\bar{x}^a)$.
1. **Initialize:** Generate an ordered partition of the conditional probability function, $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all functions whose highest variable is $X_i$. Put each observed variable in its bucket. Let $\psi_i$ be the product of input function in a bucket and let $h_i$ be the messages in the bucket.

2. **Backward:** For $p \leftarrow n$ downto 1, do
for all the functions $h_1, h_2, ..., h_j$ in $bucket_p$, do
    • **If** (observed variable) $bucket_p$ contains $X_p = x_p$, assign $X_p = x_p$ to each function and put each in appropriate bucket.
    • **else**, Generate an an $(i, m)$-partitioning, $Q' = \{Q_1, ..., Q_r\}$ of $h_1, h_2, ..., h_t$ in $bucket_p$.
    • **for each** $Q_l \in Q'$ containing $h_{l_1}, ...h_{l_t}$, **do**
          $$h_l \leftarrow max_{X_p} \prod_{j=1}^{t} h_{l_j} \qquad (8.1)$$
      Add $h_l$ to the bucket of the largest-index in $scope(h_l)$.Put constants in $bucket_1$.

3. **Forward:**
    • Generate an mpe upper bound cost by maximizing over $X_1$, the product in $bucket_1$. Namely $U \leftarrow max_{X_1} \psi_1 \prod_j h_{1_j}$.
    • (Generate an approximate mpe tuple): Given $x_1^a, ..., x_{p-1}^a$, assign $x_p^a$ to $X_p$ that maximizes the product of all functions in $bucket_p$. $L \leftarrow P(x_1^a, ..., x_n^a)$

4. **Return** $U$ and $L$ and configuration: $\bar{x}^a = (x_1^a, ..., x_n^a)$
```

Figure 8.2: Algorithm *MBE-mpe(i,m)*.

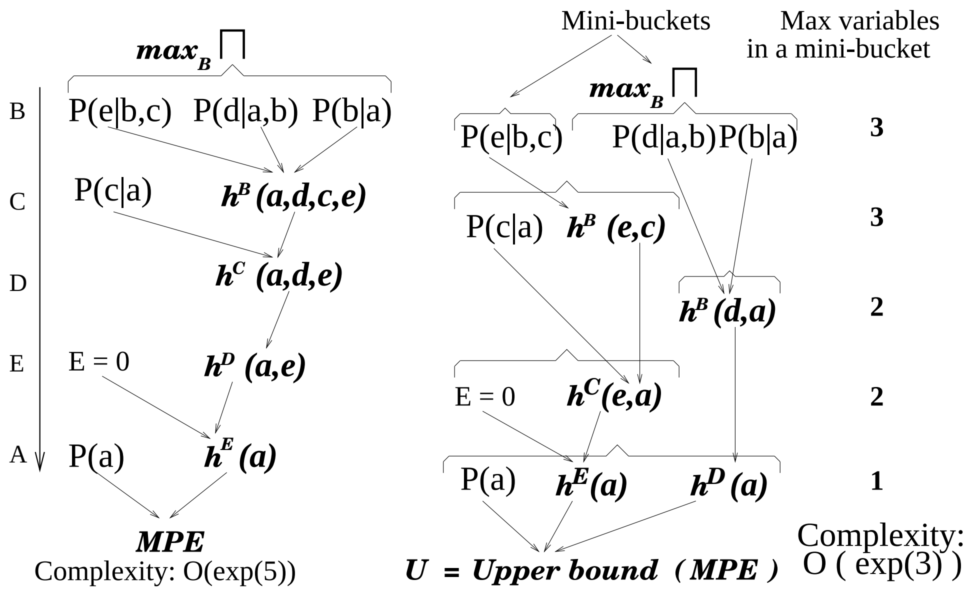


Figure 8.3: Comparison between (a) *BE-mpe* and (b) *MBE-mpe(3,2)*.

Definition 8.1.3 ((i,m)-partitioning) *A partitioning of $h_1, ..., h_t$ is* canonical *if any function f whose scope is subsumed by the scope of another function, is placed into a bucket containing 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.*

The parameters i (number of variables) and m (number of functions) are not independent, and some combinations of i and m do not yield a feasible (i,m)-partitioning. However, it is easy to see that if the i-bound on the number of variables in a mini-bucket is not smaller than the maximum scope size, then, for any value of $m > 0$, there exists an (i,m)-partitioning of each bucket. The use of the two parameters i and m, although not independent, allow considering a richer set of partitioning schemes than using i or m alone.

Clearly, since *MBE-mpe(i,m)* computes an upper bound in each bucket it yields an overall upper bound on the resulting mpe. Namely,

Theorem 8.1.4 (MBE-mpe boundness) *Algorithm* MBE-mpe*(i,m) computes an upper and a lower bounds on the mpe.*

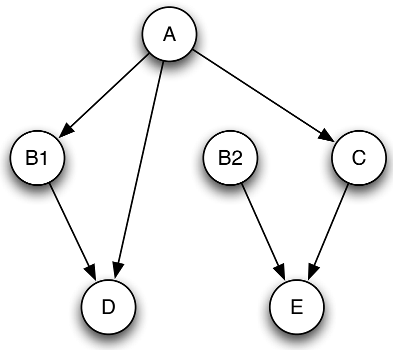
Example 8.1.5 Figure 8.3 compares algorithms *BE-mpe* and *MBE-mpe(i,m)* where $i = 3$ and $m = 2$ over the network in Figure 2.5a along the ordering $o = (A,\ E, D,\ C, B)$. The exact *BE-mpe* sequentially records the new functions (shown in boldface) $h_B(a,d,c,e)$, $h_C(a,d,e)$, $h_D(a,e)$, and $h_E(a)$. Then, in the bucket of A, it computes $M = \max_a P(a)h_E(a)$. Subsequently, an mpe configuration ($A = a'$, $B = b'$, $C = c'$, $D = d'$, $E = e'$) where $e' = 0$ is the evidence, can be computed along o by selecting a value that maximizes the product of functions in the corresponding buckets conditioned on the previously assigned values. Namely, $a' = \arg\max_a P(a)h_E(a)$, $e' = 0$, $d' = \arg\max_d h_C(a',d,e=0)$, and so on.

Looking now at *MBE-mpe(3,2)*, since bucket(B) includes five variables, we *split* it into two mini-buckets $\{P(e|b,c)\}$ and $\{P(d|a,b), P(b|a)\}$, each containing no more than 3 variables, as shown in Figure 8.3b. There can be several (3,2)-partitionings, and any choice would be legitimate, and can be selected arbitrarily. The new functions $h_B(e,c)$ and $h_B(d,a)$ are generated in different mini-buckets and are placed independently into lower buckets. In each of the remaining lower buckets that still need to be processed, the number of variables is not larger than 3 and therefore no further partitioning occurs. An upper bound on the mpe value can be computed by maximizing over A the product of functions in A's bucket: $U = max_a P(a)h_E(a)h_D(a)$.

Once all the buckets are processed, a suboptimal mpe assignment is computed by assigning a value to each variable that maximizes the product of functions in the corresponding bucket, in the same way this is done by exact *BE-mpe*. Clearly, any assignment to all the variables yields a lower bound, and one can use alternative methods to compute a configuration given the functions recorded by the mini-bucket algorithm. Note that *MBE-mpe(3,2)* does not produce functions on more than 2 variables, while the exact algorithm *BE-mpe* records a function on 4 variables. □

In summary, *MBE-mpe(i,m)* computes an interval $[L, U]$ containing the mpe value where U is the upper bound computed by the backward phase and L is the probability or cost of the returned assignment. Note however that *MBE-mpe* computes the bounds on the joint probability $mpe = \max_{\mathbf{x}} P(\mathbf{x}, \mathbf{e})$, rather than on the conditional probability $M = \max_{\mathbf{x}} P(\mathbf{x}|\mathbf{e}) = mpe/P(\mathbf{e})$. Thus

$$\frac{L}{P(\mathbf{e})} \le M \le \frac{U}{P(\mathbf{e})}$$



(a)

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

(b)

Figure 8.4: relaxed network corresponding to mini-bucket execution in Figure 8.3b

While the probability of evidence clearly influences the quality of the bound interval on M, the ratio between the upper and the lower bound is not affected.

As we will see in the next subsection, approximating posterior probabilities using bounds on joint probabilities may be more problematic.

8.1.1 The Mini-Bucket semantics

The Mini-Bucket computation can be given a useful interpretation. It can be viewed as an exact computation over a simplified graphical model where for every mini-bucket we use a new copy the bucket's variable. Namely, For each bucket and its partitioning into mini-buckets, a variable in the original problem is replaced by a set of new duplicate variables, each each associated with a single mini-bucket. For example, the Mini-Bucket trace in Figure 8.3b, corresponds to solving exactly by full bucket-elimination the network of the problem in Figure 8.4. Variable B is replaced by two copies called variables B_1 and B_2, and the functions $P(e|b,c)$, $P(d|a,b)$, and $P(b|a)$ are replaced by $P(e|b_1,c)$, $P(d|a,b_2)$ and $P(b_2|a)$. Thus the two mini-buckets correspond to two full buckets in the new simplified or relaxed problem. The relaxed problem has a smaller width and can be solved efficiently, yielding a bound (upper or lower) as expected.

Certificate of optimality. Clearly when the lower bound equals to the upper bound we have an optimal solution. Alternatively, if we use the node duplication mechanism explicitly, whenever the optimal solution of the simplified problem allow assigning the same value to duplicated variables, we know that the assignment is locally optimal, namely conditioned on the current partial configuration.

8.2 Mini-bucket approximation for belief updating

As shown in Chapter 4, the bucket elimination algorithm *BE-bel* for belief assessment is similar to *BE-mpe* except that maximization is replaced by summation and no value assignment is generated. Algorithm *BE-bel* finds $P(x_1, \mathbf{e})$ and then computes $P(x_1|\mathbf{e}) = \alpha P(x_1, \mathbf{e})$ where α is the normalization constant (see chapter 4).

The mini-bucket idea used for approximating MPE can be applied to belief updating. Let $Q\prime = \{Q_1, ..., Q_r\}$ be a partitioning of the functions $h_1, ...h_t$ in X_p's bucket. Algorithm *BE-bel* computes $h_p \leftarrow \sum_{X_p} \prod_{i=1}^{t} h_i$, over $scope(h_p) = \cup_i scope(h_i) - \{X_p\}$. (Again, we omit here the distinction between input functions and messages.) The function h_p can be rewritten as $h_p = \sum_{X_p} \prod_{l=1}^{r} \prod_{h_{l_i} \in Q_l} h_{l_i}$.

If we follow the mpe approximation precisely and migrate the summation operator into each mini-bucket, we will get $f^p_{Q\prime} = \prod_{l=1}^{r} \sum_{X_p} \prod_{l_i} h_{l_i}$. This, however, yields an unnecessarily weak upper bound of h_p where each $\prod_{l_i} h_{l_i}$ which is a function of X_p is bounded by $\sum_{X_p} \Pi_{l_i} h_{l_i}$, a constant function relative to X_p. Instead, we rewrite

$$h_p = \sum_{X_p} (\prod_{1_i} h_{1_i}) \cdot (\prod_{l=2}^{r} \prod_{l_i} h_{l_i})$$

and subsequently, instead of bounding a function of X by its sum over X, we can bound $(i > 1)$, by its maximum operator over X, yielding

$$g^p_{Q\prime} = (\sum_{X_p} \prod_{1_i} h_{1_i}) \cdot (\prod_{l=2}^{r} \max_{X_p} \prod_{l_i} h_{l_i}).$$

Therefore, an upper bound g_p of h_p can be obtained by processing one of X_p's mini-buckets by summation and the rest by maximization.

```
Algorithm MBE-bel-max(i,m)
Input: A belief network $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}_G, \prod \rangle$, an ordering $d = (\ X_1, \ldots, X_n)$ ; evidence $\mathbf{e}$
Output: an upper bound on $P(X_1, \bar{e})$ 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 $bucket_k$ Generate an $(i, m)$-partitioning,
      $Q' = \{Q_1, ..., Q_r\}$ . For each $Q_l \in Q'$, containing $h_{l_1}, ...h_{l_t}$, do
          $h_l \leftarrow \sum_{X_k} \prod_{j=1}^{t} h_{1_j},, \ if \ l = 1$
          $h_l \leftarrow max_{X_k} \prod_{j=1}^{t} h_{l_j}, \ if \ k \neq 1$
      Add $h_l$ to the bucket of the highest-index variable in its scope (and put constant functions in $bucket_1$).

3. Return $P'(\bar{x_1}, e) \leftarrow$ the product of functions in the bucket of $X_1$, which is an upper bound on $P(x_1, \mathbf{e})$.
return also $U \leftarrow \sum_{x_1} P'(x_1, \mathbf{e})$, which is an upper bound on probability of evidence.
```

Figure 8.5: Algorithm *MBE-bel-max(i,m)*.

A lower bound on the belief, or its mean value, can be obtained in a similar way. Algorithm *MBE-bel-max(i,m)* that uses the *max* elimination operator is described in Figure 8.5. Algorithms *MBE-bel-min* and *MBE-bel-mean* can be obtained by replacing the operator *max* by *min* and by *mean*, respectively. Notice however that the node duplication semantics of MBE implies summation for each of the mini-buckets. This will be remedied by an extension that generalize and improves the mini-bucket scheme, called *weighted mini-bucket.*

Theorem 8.2.1 *Given a Bayesian network with evidence e, algorithm MBE-bel-max(i,m) computes an upper bound on $P(X_1, \mathbf{e})$ and $P(\mathbf{e})$.*

We will have the same relationships between partitioning and their refinements as for the mpe case, but we have to be careful, since we have a new degree of freedom in selecting on which mini-bucket to maximize.

Proposition 8.2.2 *The following holds:*

1. *For every partitioning Q of a set of functions whose scopes include variable X_p, $h_p \leq g^p_Q \leq f^p_Q$, where f is obtained by processing each mini-bucket by summation while in g, one mini-bucket is processed by summation and the rest by maximization.*
2. *Also, if Q'' is a refinement partitioning of Q', then $h_p \leq g^p_{Q'} \leq g^p_{Q''}$.*

8.2.1 Normalization

Note that *MBE-bel-max* generates an upper bound on $P(x_1, \mathbf{e})$ but not on $P(x_1|\mathbf{e})$. If an exact value of $P(\mathbf{e})$ is not available, deriving a bound on $P(x_1|\mathbf{e})$ from a bound on $P(x_1, \mathbf{e})$ is not easy, because the normalization of upper-bounds is no an upper bound. Namely, $\frac{g(x_1)}{\sum_{x_1} g(x_1)}$, where $g(x)$ is the upper bound on $P(x_1, \mathbf{e})$, is not necessarily an upper bound on $P(x_1|\mathbf{e})$. As noted we can derive a lower bound, f, on $P(\mathbf{e})$ using *mbe-bel-min* and then compute $\frac{g(x_1)}{f}$ as an upper bound on $P(x_1|\mathbf{e})$. This however is likely to generate weak bounds due to compounded error.

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

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

Therefore, although $P(\mathbf{e})$ is not known, the ratio of upper to lower bounds remains constant. Yet, the difference between the upper and the lower bounds can grow substantially, especially in cases of rare evidence. Note that if $P(\mathbf{e}) \leq U_i$, we get $\frac{L_i}{P(\bar{e})} \leq P(X_1|\bar{e}) \leq 1$, yielding a trivial upper bound. Finally, no guarantee is given for $g_{mean}(x_i)$, and therefore, the approximation of $\frac{g_{mean}(x_i)}{\sum_{x_1} g_{mean}(x_1)}$ can be below or above the exact value. Interestingly, the computation of $\frac{g_{mean}(X_1=x_i)}{\sum_{x_1} g_{mean}(x_1)}$ is automatically achieved when processing all the mini-buckets by summations, and subsequently normalizing. Namely, this is what is obtained when we transform the original network by node duplication and apply exact BE-bel to the simplified network.

8.3 Weighted Mini-Bucket Elimination

The power sum is defined as follows:

$$\sum_x^w f(x) = (\sum_x f(x)^{\frac{1}{w}})^w \tag{8.2}$$

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

Proposition 8.3.1 (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,*

$$\sum_x^w \prod_{i=1}^r f_i(x) \leq \prod_{i=1}^r \sum_x^{w_i} f_i(x)$$

This means that if we generalize to have a power-sum-product query over our graphical model relative to a given weight w, the bucket elimination immediately applies to yield an exact algorithm, when using $\otimes = \sum_x^w$. Most significantly, due to Holder inequality we get a *weighted*-mini-bucket algorithm which is correct for any set of weights satisfying EQ. (8.2). The case of w=1 specialize to summation and w=0 to maximization. When $w = 0$, the only consistent weight vector is $w_i = 0$, which means that each mini-bucket should be processed by maximization. But, when $w = 1$, namely for the summation query, an appropriate choice of the mini-bucket weights can tighten the obtained bound. We can try to select a good, or even optimal set of weights for each bucket to yield the tightest bounds, at least in principle. The max-sum case that we proposed in MBE-max-bel, correspond to assigning one mini-bucket with $w = 1$ and the rest with $w = 0$.

Algorithm weighted WMBE is given in Figure 8.6. It is paramaterized by a set of n weights, one for each variable, so it can accomodate a variety of tasks. When we solve a pure summation task such as the probability of evidence or the posterior probability the weight for each variable is $w = 1$. For pure optimization the weights are all $w = 0$. But, as we will see next, for mixed max-sum product queries such as marginal maps (mmap), we can specialize weights based on variables, so that buckets of sum variables will be assigned $w = 1$ and others $w = 0$ for maximization. This yields an elegant framework expressing the mini-bucket algorithms and permits the additional control via the spliting weights for each mini-bucket of a summation variable.

8.4 Mini-bucket elimination for MAP

Algorithm *BE-map* is a combination of *BE-mpe* and *BE-bel* as we have shown in Chapter **??**; some of the variables are eliminated by summation, while the others by maximization.

Given a belief network, a subset of hypothesis variables $A = \{A_1, ..., A_k\}$, and evidence $\mathbf{e}$, the problem is to find an assignment to the hypothesized variables that maximizes their probability conditioned on $\mathbf{e}$. Formally, we wish to find

$$\bar{a}_k^{map} = \arg\max_{\bar{a}_k} P(\bar{a}_k|\mathbf{e}) = \arg\max_{\bar{a}_k} \frac{\sum_{\mathbf{x}_{(k+1)..n}} \prod_{i=1}^{n} P(x_i, \mathbf{e}|\mathbf{x}_{pa_i})}{P(\mathbf{e})} \tag{8.3}$$

where $\mathbf{x} = (a_1, ..., a_k, x_{k+1}, ..., x_n)$ denotes an assignment to all variables, while $\bar{a}_k = (a_1, ..., a_k)$ and $\mathbf{x}_{(k+1)..n} = (x_{k+1}, ..., x_n)$ denote assignments to the hypothesis and non-hypothesis variables, respectively. Since $P(\mathbf{e})$ is a normalization constant, the same maximum is obtained for $P(\bar{a}_k, \mathbf{e})$.

[I temporarily provide both descriptions of MBE-map]

As we know, the bucket-elimination algorithm for finding the exact map, *BE-map*, assumes only orderings in which the hypothesized variables appear first and thus are processed last by the algorithm. This restriction makes the task more difficult because

```
**Algorithm Weighted WMBE(i,m),** $(w_1, ...w_n)$
**Input:** A belief network $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}_G, \prod \rangle$, an ordering $d = (\ X_1, \ldots, X_n)$ ; evidence $\mathbf{e}$
**Output:** an upper bound on $\sum_{\mathbf{X}}^{w} \prod_{i=1}^{n} P_i$
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 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 $bucket_1$).

   • **Else** for $h_1, h_2, ..., h_t$ in $bucket_k$, generate an $(i, m)$-partitioning,
     $Q^{'} = \{Q_1, ..., Q_r\}$ . Select a set of weights $w_1, ...w_r$ s.t $\sum_l w_l = w.$, where
     $w$ is the weight of the bucket
     **For each** $Q_l \in Q^{'}$, containing $h_{l_1}, ...h_{l_t}$, do

        $$h_l \leftarrow \sum_{X_k}^{w_l} \prod_{j=1}^{t} h_{l_j} = (\sum_{X_k} \prod_{j=1}^{t} (h_{l_j})^{w_l})^{\frac{1}{w_l}}$$

     Add $h_l$ to the bucket of the highest-index variable in its scope.

3. **Return** $U \leftarrow$ the power weighted sum of the product of functions in the bucket
   of $X_1$, which is an upper bound on $\sum_{\mathbf{X}}^{w} \prod_{i=1}^{n} P_i(\cdot|\mathbf{e})$.
```

Figure 8.6: Algorithm *WMBE(i,m)*.

```
Algorithm MBE-map(i,m)
Input: A Bayesian network $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}_G, \prod \rangle$, $P = \{P_1, ..., P_n\}$; a subset of
hypothesis variables $A = \{A_1, ..., A_k\}$; an ordering of the variables, $d$, in which
the $A$'s are first in the ordering; observations $\mathbf{e}$.
Output: An upper bound on the map and a and a suboptimal solution $A = \bar{a}^a_k$.
1. Initialize: Partition $P = \{P_1, ..., P_n\}$ into $bucket_1, \ldots, bucket_n$, where $bucket_i$
contains all functions whose highest variable is $X_i$.
2. Backwards For $p \leftarrow n$ downto 1, do
for all the functions $h_1, h_2, ..., h_j$ in $bucket_p$ do
    • If (observed variable) $bucket_p$ contains the observation $X_p = x_p$, assign
      $X_p = x_p$ to each $h_i$ and and put each in appropriate bucket.
    • Else for $h_1, h_2, ..., h_j$ in $bucket_p$ generate an $(i, m)$-partitioning, $Q' =$
      $\{Q_1, ..., Q_r\}$.
    • If $X_P \notin A$ assign $w_p = 1$, otherwise $w_p = 0$. Select weights for the
      mini-buckets in $X_p$ bucket: $w_{p_1}, ..., w_{p_r}$. s.t $\sum_i w_{p_i} = w_p$.
      foreach $Q_l \in Q'$, containing $h_{l_1}, ...h_{l_t}$, do

      $$h_l \leftarrow \sum_{X_k}^{w_{p_l}} \prod_{j=1}^{t} h_{l_j} = (\sum_{X_k} (\prod_{j=1}^{t} h_{l_j})^{w_{p_l}})^{\frac{1}{w_{p_l}}}$$

      Add $h_l$ to the bucket of the highest-index variable in its scope.
3. Forward: for $p = 1$ to $k$, given $A_1 = a^a_1, ..., A_{p-1} = a^a_{p-1}$, assign a value $a^a_p$ to
$A_p$ that maximizes the product of all functions in $bucket_p$. conditioned on earlier
assignments.
4. Return An upper bound $U = max_{a_1} \prod_{h_i \in bucket_1} h_i$ on the $map$ value, com-
puted in the first bucket, and the assignment $\bar{a}^a_k = (a^a_1, ..., a^a_k)$.
```

Figure 8.7: Algorithm *MBE-map(i,m)*.

```
Algorithm MBE-map(i,m)
Input: A Bayesian network $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}_G, \prod \rangle$, $P = \{P_1, ..., P_n\}$; a subset of
hypothesis variables $A = \{A_1, ..., A_k\}$; an ordering of the variables, $d$, in which
the $A$'s are first in the ordering; observations $e$. $\psi_i$ is the product of input func-
tion in the bucket of $X_i$.
Output: An upper bound on the map and a and a suboptimal solution $A = \bar{a}^a_k$.
1. Initialize: Generate an ordered partition of the conditional probability func-
tions, $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all functions whose highest
variable is $X_i$.
2. Backwards For $p \leftarrow n$ downto 1, do
for all the message functions $\beta_1, \beta_2, ..., \beta_j$ in $bucket_p$ do
    • If (observed variable) $bucket_p$ contains the observation $X_p = x_p$, assign
      $X_p = x_p$ to each $\beta_i$ and $\psi_p$ and put each in appropriate bucket.
    • Else for $h_1, h_2, ..., h_j$ in $bucket_p$ generate an $(i, m)$-partitioning, $Q^{'}$ mini-
      buckets $Q_1, ..., Q_r$.
    • If $X_P \notin A$ (not a hypothesis variable)
      foreach $Q_l \in Q^{'}$, containing $h_{l_1}, ...h_{l_t}$, do
          $h_l \leftarrow \sum_{X_p} \Pi_{i=1}^t h_{1_i},\ if\ l = 1$
          $h_l \leftarrow max_{X_p} \Pi_{i=1}^t h_{l_i},\ if\ l \neq 1$
    • Else, $(X_p \in A)$ ( a hypothesis variable)
      for each $Q_l \in Q^{'}$ containing $h_{l_1}, ...h_{l_t}$
          $h_l \leftarrow max_{X_p} \Pi_{i=1}^t h_{l_i}$
      Add $h^l$ to the bucket of the highest-index variable in $U_l \leftarrow \bigcup_{i=1}^t scope(h_{l_i}) -$
      $\{X_p\}$, (put constants in $bucket_1$).
3. Forward: for $p = 1$ to $k$, given $A_1 = a^a_1, ..., A_{p-1} = a^a_{p-1}$, assign a value $a^a_p$ to
$A_p$ that maximizes the product of all functions in $bucket_p$.
4. Return An upper bound $U = max_{a_1} \prod_{h_i \in bucket_1} h_i$ on $map$, computed in the
first bucket, and the assignment $\bar{a}^a_k = (a^a_1, ..., a^a_k)$.
```

Figure 8.8: Algorithm *MBE-map(i,m)*.

it implies higher induced widths. The algorithm has the usual backward phase and its forward phase however is relative to the hypothesis variables only.

The mini-bucket scheme for map is a straightforward extension of the mini-bucket algorithms for summation and maximization and easy to express using the weighted mini-bucket scheme. As usual, we partition each bucket into mini-buckets as before. If the bucket's variable can be eliminated by summation, we apply the power-sum with $w = 1$. The rest of the buckets are processed by the mini-bucket rule with $w = 0$, namely by maximization. Namely, when the algorithm reaches the hypothesis buckets, their processing is identical to that of *MBE-mpe*. Algorithm *MBE-map(i,m)* is described in Figure **??**.

Decoding the map assignment and the issue with lower-bounds. Once the backwards pass of *MBE-map* terminates, we have an upper bound and, in principle, we can compute a map assignment in the forward phase. While the probability of this assignment is a lower bound for the map value, computing the actual probability is no longer a simple forward step over the generated buckets but requires an exact inference. We cannot use the functions generated by *MBE-bel-max* in the buckets of summation variables since those serve as upper bounds. One possibility is, once an assignment is obtained, to rerun the mini-bucket algorithm over the non-hypothesis variables using the min operator (as in *MBE-bel-min*, and then compute a lower bound on the assigned tuple in another forward step over the first k buckets. We will leave the details of this idea as an exercise.

Example 8.4.1 Consider a belief network which describes the decoding of a *linear block code*, shown in Figure 8.9. 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 $Y = (\bar{y}^u, \bar{y}^x)$, which is the map task where U is the set of hypothesis variables, and $Y = (\bar{y}^u, \bar{y}^x)$ is the evidence. After processing the observed buckets we get the following bucket configuration (lower case y's are observed values):
$bucket(X_0) = P(y_0^x|X_0), P(X_0|U_0, U_1, U_2)$,
$bucket(X_1) = P(y_1^x|X_1), P(X_1|U_1, U_2, U_3)$,
$bucket(X_2) = P(y_2^x|X_2), P(X_2|U_2, U_3, U_4)$,

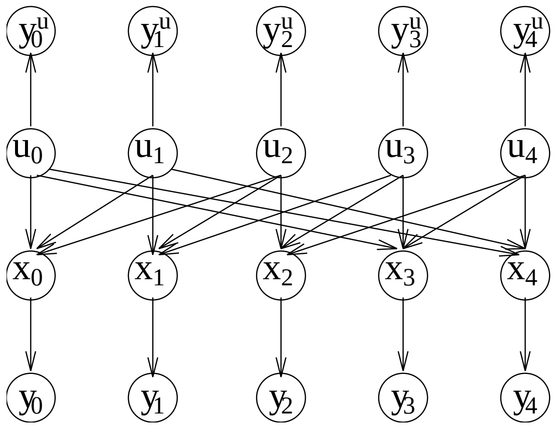


Figure 8.9: Belief network for a linear block code.

$bucket(X_3) = P(y_3^x|X_3), P(X_3|U_3, U_4, U_0),$
$bucket(X_4) = P(y_4^x|X_4), P(X_4|U_4, U_0, U_1),$
$bucket(U_0) = P(U_0), P(y_0^u|U_0),$
$bucket(U_1) = P(U_1), P(y_1^u|U_1),$
$bucket(U_2) = P(U_2), P(y_2^u|U_2),$
$bucket(U_3) = P(U_3), P(y_3^u|U_3),$
$bucket(U_4) = P(U_4), P(y_4^u|U_4).$

Processing by *MBE-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:
$bucket(X_0) = \{P(y_0^x|X_0), P(X_0|U_0, U_1, U_2)\},$
$bucket(X_1) = \{P(y_1^x|X_1), P(X_1|U_1, U_2, U_3)\},$
$bucket(X_2) = \{P(y_2^x|X_2), P(X_2|U_2, U_3, U_4)\},$
$bucket(X_3) = \{P(y_3^x|X_3), P(X_3|U_3, U_4, U_0)\},$
$bucket(X_4) = \{P(y_4^x|X_4), P(X_4|U_4, U_0, U_1)\},$
$bucket(U_0) = \{P(U_0), P(y_0^u|U_0), h^{X_0}(U_0, U_1, U_2)\}, \{h^{X_3}(U_3, U_4, U_0)\}, \{h^{x_4}(U_4, U_0, U_1)\},$
$bucket(U_1) = \{P(U_1), P(y_1^u|U_1), h^{x_1}(U_1, U_2, U_3), h^{U_0}(U_1, U_2)\}, \{h^{u_0}(U_4, U_1)\},$
$bucket(U_2) = \{P(U_2), P(y_2^u|U_2), h^{x_2}(U_2, U_3, U_4), h^{U_1}(U_2, U_3)\},$
$bucket(U_3) = \{P(U_3), P(y_3^u|U_3), h^{u_0}(U_3, U_4), h^{U_1}(U_3, U_4), h^{u_2}(U_3, U_4)\},$
$bucket(U_4) = \{P(U_4), P(y_4^u|U_4), h^{u_1}(U_4), h^{u_3}(U_4)\}.$
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 *Upper* satisfying $Upper \geq map = P(U, \bar{y}^u, \bar{y}^x)$ where $\bar{y}^u$ and , $\bar{y}^x$ are the observed outputs for the U's and the X's bits transmitted. In order to bound $P(U|\bar{e})$, where $\bar{e} = (\bar{y}^u, \bar{y}^x)$, we need $P(\mathbf{e})$ which is not available. Yet, again, in most cases we are interested in the ratio $P(U = \bar{u_1}|\mathbf{e})/P(U = \bar{u_2}|\mathbf{e})$ for competing hypotheses $U = \bar{u_1}$ and $U = \bar{u_2}$ rather than in the absolute values. Since $P(U|\mathbf{e}) = P(U, \mathbf{e})/P(\mathbf{e})$ and the probability of the evidence is just a constant factor independent of U, the ratio is equal to $P(U_1, \mathbf{e})/P(U_2, \mathbf{e})$. $\square$

Exercise: What is the relaxed network that corresponds to the above computation? How would you generate a candidate map assignment? how would you compute its probability? What is the relationship between the map and the mpe assignments?

8.5 Mini-buckets for general discrete optimization; The min-sum query

The mini-bucket principle can also be applied to deterministic discrete optimization problems which can be defined over *cost networks*, yielding approximation to dynamic programming for discrete optimization [13]. In fact, as we showed (Chapter 2) the mpe task is a special case of combinatorial optimization and its approximation via mini-buckets can be straightforwardly extended to the general case. For completeness we present the algorithm explicitly for cost networks, namely, for the min-sum problem.

As defined earlier a *cost network* is a triplet (X, D, C), where X is a set of discrete variables, $X = \{X_1, ..., X_n\}$, over domains $D = \{D_1, ..., D_n\}$, and C is a set of real-valued cost functions $C_1, ..., C_l$. The *cost function* is defined by $C(X) = \sum_{i=1}^{l} C_i$. The optimization (minimization) problem is to find an assignment $x^{opt} = ({x_1}^{opt}, ..., {x_n}^{opt})$ such that $C(x^{opt}) = \min_{x=(x_1,...,x_n)} C(x)$.

Algorithm *mbe-opt* is described for the sake of completeness in Figure 8.10. Step 2 (backward step) computes a lower bound on the cost function while Step 3 (forward step) generates a suboptimal solution which provides an upper bound on the cost function.

Unified presentation of MBE. Clearly the mini-bucket scheme is applicable to any bucket-elimination scheme and can be described within the general framework using the combination and marginalization operators. The power-sum provide one way to generalize this scheme, which is called Weighted mini-bucket elimination. [?]. (Exercise: Provide a general description of the mini-bucket elimination scheme using the combination and marginalization operators.)

8.6 Complexity and tractability

8.6.1 The case of low induced width

We denote by *mini-bucket-elimination(i,m)*, or simply *MBE(i,m)*, a generic mini-bucket scheme with parameters i and m, without specifying the particular task it solves.

It is easy to derive *MBE(i,m)* complexity as the bucket-elimination complexity of the relaxed problem generated by variable duplication. Since variable duplication can generate at most r additional variables, where r is the number of functions, but will leave the number of functions fixed at r, and since the resulting problem has induced-width bounded by i, *MBE's* complexity on such a problem is derived from BE's complexity:

Theorem 8.6.1 *Algorithm* MBE(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*[1]*. 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, $|S| \leq i \leq n$.*

We can associate a bucket-elimination or a mini-bucket elimination algorithm with a *computation tree* where leaf nodes correspond to the original input functions (CPTs or cost functions), and each internal node v corresponds to the result of applying an elimination operator (e.g., product followed by summation) to the set of node's children, $ch(v)$ (children correspond to all functions in the corresponding bucket or mini-bucket).

[1]Note that $r = n$ for Bayesian networks, but can be higher or lower for general constraint optimization tasks

```
Algorithm MBE-opt(i,m)
Input: A cost network $(X, D, C)$, $C = \{C_1, ..., C_l\}$; ordering $o$, a set of assignments $e$.
Output: A lower and an upper bound on the optimal cost.
1.Initialize: Partition $C$ into $bucket_1$, ..., $bucket_n$, where $bucket_p$ contains all components $h_1, h_2, ..., h_t$ whose highest-index variable is $X_p$.
2. Backward: for $p = n$ to 2 do
    • If $X_p$ is observed ($X_p = a$), replace $X_p$ by $a$ in each $h_i$ and put the result in its highest-variable bucket (put constants in $bucket_1$).
    • Else for $h_1, h_2, ..., h_t$ in $bucket_p$ generate an $(i, m)$-partitioning,
      $Q' = \{Q_1, ..., Q_r\}$.
    • For each $Q_l \in Q'$ containing $h_{l_1}, ...h_{l_t}$,
          $h_l \leftarrow min_{X_p} \sum_{i=1}^{t} h_{l_i}$
      and add it to the bucket of the highest-index variable in its scope. (put constants in $bucket_1$).
3. Forward: for $p = 1$ to $n$, given $X_1 = x_1^{opt}, ..., X_{p-1} = x_{p-1}^{opt}$,
   assign a value $x_p^{opt}$ to $X_p$ that minimizes the sum of all functions in $bucket_p$.
4. Return the assignment $x^{opt} = (x_1^{opt}, ..., x_n^{opt})$, an upper bound $U = C(x^{opt})$, and a lower bound $L = min_{x_1} \sum_{h_i \in bucket_1} h_i$ on the optimal cost.
```

Figure 8.10: Algorithm *MBE-opt(i,m)*.

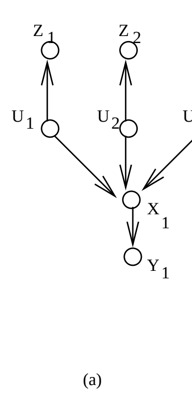
We can compress the computation tree so that each node having a single child will be merged into one node with its parent, so that the branching degree in the resulting tree is not less than 2. Computing an internal node that is a compressed sequence of single-child nodes takes $O(exp(|S|)$ time and space since it only requires a sequence of elimination operations over a single function which can be accomplished in one pass through the tuples (accumulating appropriate running summations over the relevant variables). The cost of computing any other internal node v is $O(|ch(v)| \cdot exp(i))$ where $|ch(v)| \leq m$ and i bounds the resulting scope size of generated functions. Since the number of leaf nodes is bounded by r, the number of internal nodes in the computation tree is bounded by r as well (since the branching factor of each internal node is at least 2). Thus the total amount of computation over all internal nodes in the computation tree is time and space $O(r \cdot exp(i))$ in general, which becomes to $O(n \cdot exp(i))$ for belief networks.

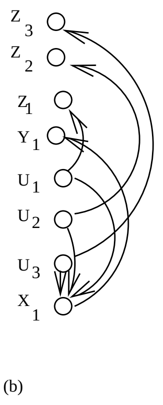
Clearly, when the induced-width along the processing order is smaller than i, MBE(i, n) coincides with bucket-elimination and is therefore exact. This is because each full bucket satisfies the condition of being an (i, n)-partitioning.

Interestingly, algorithm *MBE(i,m=1)* is exact for acyclic networks, and in particular for polytrees, if applied along a proper orderings. Such orderings are determined by consulting a rooted join-tree of the acyclic network (we know that such a join-tree exists from the definition of an acyclic network as discussed in Chapter 5). We then order the variables from last to first by selecting and removing a leaf function from the rooted join-tree and placing as next all its variables that were not ordered yet. This way, each bucket would contain at most one non-subsumed function. Thus,

Theorem 8.6.2 *Given an acyclic network, there exists an ordering such that algorithm* MBE(n,1) *is exact and is time and space $O(n \cdot exp(|S|))$, where $|S|$ is the largest scope size of any input function.*

Example 8.6.3 Consider an ordering $o = (X_1, U_3, U_2, U_1, Y_1, Z_1, Z_2, Z_3)$ of the polytree in Figure 8.11a, where the last four variables $Y_1,\ Z_1,\ Z_2,\ Z_3$ in the ordering are observed. Once the last four buckets are processed as observation buckets, we get (observed values shown in low-case):
$bucket(U_1) = P(U_1), P(X_1|U_1, U_2, U_3), P(z_1|U_1),$
$bucket(U_2) = P(U_2), P(z_2|U_2),$





(a) (b)

Figure 8.11: (a) A polytree and (b) a legal ordering, assuming that nodes Z_1, Z_2, Z_3 and Y_1 are observed.

$bucket(U_3) = P(U_3), P(z_3|U_3)$
$bucket(X_1) = P(y_1|X_1)$.
We can see that when $m = 1$ the algorithm will have no partitioning in any bucket, because each bucket has at most one non-subsumed function.

□

8.6.2 Heuristic for mini-bucket partitionings

Clearly, there are many ways to partition a bucket into a collection of mini-buckets having parameters i and m. These partitionings can be guided by the graph, or by the content of the actual functions, aiming to minimize the error incurred by a particular partition. We note that in practice the mini-bucket scheme is used primarily with the i-bound and ignores the m-bound. Namely, the mini-bucket is bounded by the number of variables it contain no matter how many functions it has. This is because controling the mini-bucket size by the number of variables is more refined. One popular partitioning heuristic, is scope-based, relying solely on the scopes of the functions. We refer to the procedure that takes a bucket B and partition it into mini-buckets having at most i variables as *partitioning(B,i)*. It outputs an i-partition.

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. This is equivalent to minimizing the number of copies of each variable. According to this scheme, first, single function mini-buckets are decreasingly ordered according to their arity. Then, each mini-bucket, from first to last, is absorbed into the earliest mini-bucket with whom it can be merged. The time and space complexity of `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, where $|B|$ is the number of variables in the bucket. (Exercise: prove this complexity).

The scope-based heuristic can be computed quickly, but its shortcoming is that it does not consider the actual information contained in each function. Bucket partitioning strategies that take into account the functions themselves were also explored. Given a bucket B, the goal of the partition process is to find an i-partition Q of B such that the function computed by the collection of mini-buckets g_Q is the *closest* to the exact bucket function g, according to some distance measure *dist*. Thus, the partition task is to find an i-partition Q^* of B such that $Q^* = \arg\min_Q dist(g_Q, g)$. The distance measures considered included *log relative error*, *maximum log relative error*, *KL divergence* and *absolute error*. For example:

- *Log relative error*:

$$RE(f, h) = \sum_t (\log(f(t)) - \log(h(t)))$$

- *Max log relative error*:

$$MRE(f, h) = \max_t \{\log(f(t)) - \log(h(t))\}$$

We can organize the space of partitions in a lattice using the *refinement* relation since it yields a partial order. Each partition Q of bucket B is a vertex in the lattice. There is an upward edge from Q to Q' if Q' results from merging two mini-buckets of Q in which case Q' is a *child* of Q. The set of all children of Q is denoted by $ch(Q)$. The *bottom* partition in the lattice is $Q^{\perp}$ while the *top* partition is $Q^{\top}$. For any two partitions Q and Q', if Q' is a descendent of Q then $g^{Q'}$ is clearly tighter than g^{Q} .

Example 8.6.4 Consider a bucket $\mathcal{B}_x = \{f_1, f_2, f_3, f_4\}$. Its Hasse diagram in depicted in Figure 8.12. As observed, the finest partition is $Q^{\perp} = \{\{f_1\}, \{f_2\}, \{f_3\}, \{f_4\}\}$ (depicted

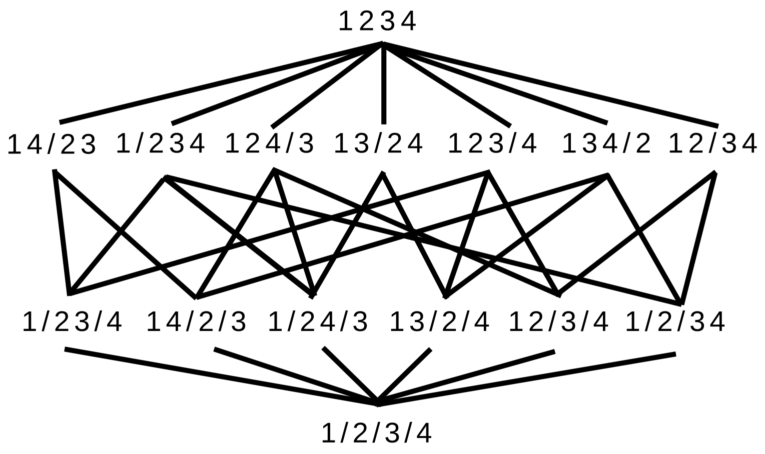


Figure 8.12: Partitioning lattice of bucket $\{f_1, f_2, f_3, f_4\}$. We specify each function by its subindex.

in the bottom of the diagram). The coarsest partition is $Q^{\top} = \{\{f_1, f_2, f_3, f_4\}\}$ (depicted in the top of the diagram). $\square$

Since an optimal partition-seeking algorithm may need to traverse the partitioning lattice bottom-up along all paths, yielding a computationally hard task, only depth-first greedy traversals schemes may be considered. A guiding heuristic function along the lattice can be constructed based on the error that occur when combining two functions into a single mini-bucket versus keeping them separate,

The traversal can be guided by a heuristic function h defined for a partition Q and its child partition Q', denoted $Q \to Q'$. The local distance heuristics derived from the above distance measures yield *content-based* local partitioning heuristics(see [86]). At each step, the algorithm ranks each child Q' of the current partition Q according to such an h. Clearly, each iteration is guaranteed to tighten the resulting bound.Algorithm GreedyPartition is given in Figure 8.13.

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.*

```
**function** GreedyPartition($\mathcal{B}$,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

8.6.3 Generalized MBE

The schedule by which mini-buckets are identified and processed can be relaxed. It does not have to be regimented to be processed in blocks for each variable. In other words, we can process a just single mini-bucket of B first, yielding a new, relaxed problem to which we can apply the mini-bucket recursively. In particular we can identify, and process a mini-bucket of C and then go back and process another mini-bucket of B and so on. This alternative schedule is apparent once we reason about partitioning heuristics as variable duplications. Finally, a relaxed network due to variable-duplication can be processed by any means, not necessarily by bucket-elimination. This observation open ups a richer collection of bounding schemes that can be considered. For more (see Larkin)

8.7 Using the mini-bucket as an anytime scheme

The mini-bucket scheme can be used as a stand alone approximation. Yet it can be extended into an anytime scheme or can be augmented within a search scheme as described next.

An important is that the scheme provides an adjustable trade-off between accuracy of a solution and the complexity of deriving it. Both the accuracy and the complexity increase monotonically with the parameters i and m. While in general it may not be easy to predict the algorithm's performance for a particular parameter setting, it is possible to use this scheme within an *anytime* framework. *Anytime algorithms* can be interrupted at any time producing the best solution found thus far. As more time is available, better solutions will be generated.

```
Algorithm ANYTIME-mpe($\epsilon$)
Input: A belief network $\mathcal{B} =< X, D, G, \mathcal{P} >$, where $\mathcal{P} = \{P_1, ..., P_n\}$;
an ordering of the variables, $d = X_1, ..., X_n$; observations $\mathbf{e}$.
Initial values, $i_0$ and $m_0$; increments $i_{step}$ and $m_{step}$, approximation error $\epsilon$.
Output: An upper bound $U$ and a lower bound $L$ on the $MPE = \max_{\bar{x}} P(\bar{x}, \bar{e})$,
and a suboptimal solution $\bar{x}^a$ that provides a lower bound $L = P(\bar{x}^a)$.
1. Initialization: $i = i_0, m = m_0$.
2. while resources are available, do
    • run MBE-mpe(i,m)
    • $U \leftarrow$ upper bound of MBE-mpe(i,m)
    • $L \leftarrow$ lower bound of MBE-mpe(i,m)
    • Retain best bounds $U, L$, and best solution found so far
    • if $1 \leq U/L \leq 1 + \epsilon$, return solution
    • else increase $i$ and $m$: $i \leftarrow i + i_{step}$ and $m \leftarrow m + m_{step}$
3. Return the largest $L$ and the smallest $U$ found so far.
Return the corresponding mpe assignment.
```

Figure 8.14: Algorithm *ANYTIME-mpe(ϵ)*.

We can have an anytime algorithm by running a sequence of mini-bucket algorithms with increasing values of i and m until either a desired level of accuracy is obtained, or until the computational resources are exhausted. To illustrate, such an anytime scheme for finding the mpe, *ANYTIME-mpe(ϵ)* is presented in Figure 8.14, where using a parameter ϵ to express the desired accuracy level. The algorithm uses initial parameter settings, i_0 and m_0, and increments i_{step} and m_{step}. *MBE-mpe(i,m)* computes a suboptimal *mpe* solution and the corresponding lower and upper bounds L, and U for increasing values of i and m. The algorithm terminates when either $1 \leq U/L \leq 1 + \epsilon$, or when the computational resources are exhausted, returning the largest lower bound and the smallest upper bound found so far, as well as the current highest suboptimal solution.

For other queries, such as belief computations deriving the upper and lower bound can be done using two runs of the algorithm (with the max and the min operators, respectively).

The above scheme can be refined and improved to save redundant computation in the repeated mini-bucket runs and also in controlling the partitioning from one run to the next in a manner that will guarantee improvements by combining some mini-buckets, from one iteration to the next. (Exercise: design an anytime scheme that allows computation sharing between subsequent mbe processing.)

Embedding mini-bucket as heuristic generation for search. Another orthogonal anytime extension of the mini-bucket, is to embed it within a complete anytime heuristic search algorithm such as *branch-and-bound* for optimization tasks. Since, the mini-bucket scheme computes bounds (upper or lower) on the exact quantities, these bounds can be used as heuristic functions to guide search algorithms and for pruning the search space. In other words, rather than stopping with the first solution found (which is a lower bound), as it is done in the forward step of *MBE-mpe*, we can continue searching for better solutions, while using the mini-bucket functions to guide and prune the search. This approach was explored extensively in recent year yielding state-of the art algorithms both for finding an mpe assignment as well as for constraint optimization problems [56, 66].

In the context of sum-product queries such as belief updating and probability of evidence, the mini-bucket's output can be viewed as approximating the probability distribution of the Bayesian network. This can be used as a basis for sampling (e.g., importance sampling) and for guiding search to bound quantities of interest. More on this in future chapters. For details on this direction see [81, 98, 49, 48].

8.8 From mini-bucket to mini-clustering

The mini-bucket idea can be extended to any tree-decomposition scheme. In this section we will describe one such extension called *Mini-Clustering (MC)*. The benefit of this algorithm is that all single-variable beliefs are computed (approximately) at once, using a two-phase message-passing process along the cluster tree like in the mini-bucket bounded inference.

We focus on likelihood computations (belief-updating and probability of evidence) for which such extensions (from mini-bucket to mini-clustering) are most relevant. We will consider a general belief network $BN =< X, D, G, P >$ and its *tree-decomposition* defined as usual by $< T, \chi, \psi >$, where $T = (V, E)$ is a tree, and χ and ψ are the labeling functions which associate with each vertex $v \in V$ two sets, $\chi(v) \subseteq X$ and $\psi(v) \subseteq P$.

Rather than computing the mini-bucket approximation n times, one for each variable as would be required by the mini-bucket approach, *mini-clustering* performs an equivalent computation with just two message passings along each arc of the tree-decomposition. Remember that a tree-decomposition assigns to each node u in the tree T a set of variables $\chi(u)$ and a set of functions $\psi(u)$ (see Chapter 5. During CTE's processing (see Algorithm 5.8) of a cluster u it contains the original functions as well as messages from neighboring clusters which we denote as $cluster(u)$. We can partition $cluster(u)$ into p mini-clusters $mc(1), \ldots, mc(p)$, each having at most i variables, where i is the i-bound controlling the accuracy. Instead of computing the message $h_{u \to v} = \sum_{elim(u,v)} \prod_{f \in cluster(u)} f$ by CTE-bel; we can divide the functions of $cluster(u)$ into p mini-clusters and rewrite $h_{u \to v} = \sum_{elim(u,v)} \prod_{f \in cluster(u)} f = \sum_{elim(u,v)} \prod_{k=1}^{p} \prod_{f \in mc(k)} f$. By migrating the summation operator into each mini-cluster, yielding $\prod_{k=1}^{p} \sum_{elim(u,v)} \prod_{f \in mc(k)} f$, we get an upper bound on $h_{u \to v}$. The resulting algorithm, called MC-bel(i), (we drop the m parameter for simplicity), is presented in Figure 8.15.

The combined functions are approximated via mini-clusters, as follows. Suppose $u \in V$ has received messages from all its neighbors other than v (the message from v is ignored even if received). The functions in $cluster_v(u)$ that are to be combined are partitioned into mini-clusters $\{mc(1), \ldots, mc(p)\}$, each one containing at most i variables. Each mini-cluster is processed by summation over the eliminator, or by maximization, and the resulting combined functions as well as all the individual functions (not dependant on the separator) are sent to v.

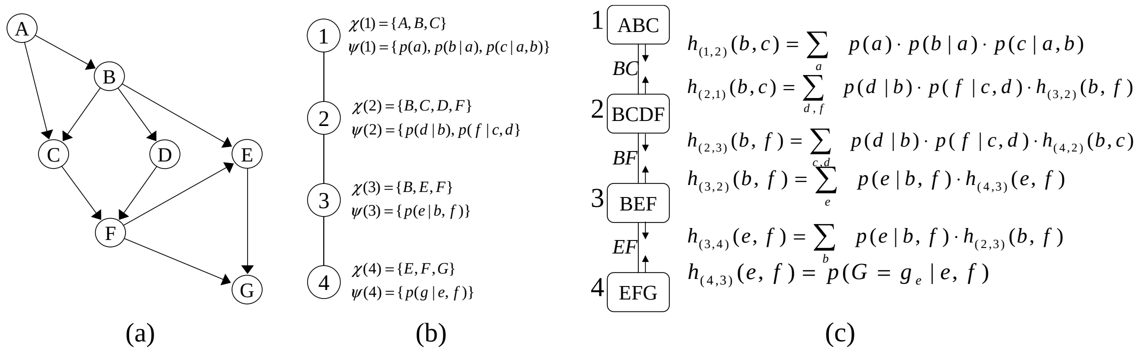
As in the mini-bucket case we can also derive a lower-bound on beliefs by replacing the *max* operator with *min* operator. This allows computing both an upper bound and a lower bound on the joint beliefs. Alternatively, if we forgo the desire to derive a bound, we can replace *max* by a *mean* operator (taking the sum and dividing by the number of elements in the sum), deriving an approximation of the joint belief.

Mini-Clustering Elimination for Belief Updating (MC-bel-max(i))
Input: A tree decomposition $\langle T, \chi, \psi \rangle T = (V, E)$ for
$\mathcal{B} = \langle X, D, G, P \rangle$. Evidence variables $var(e)$. An i-bound parameter i.
Output: An augmented tree whose nodes are clusters containing the original CPTs as well as messages received from neighbors. An upper bound for $P(X_i, e)$.
Denote by $H_{u \to v}$ the message sent by vertex u to vertex v, $ne_v(u)$ the neighbors of u in T excluding v.
$cluster(u) = \psi(u) \cup \{H_{v \to u} | (v, u) \in E\}$.
1. **Compute the combined mini-functions: For** every node u in the cluster tree, T, once u has received messages from all $ne_v(u)$, compute message to node v:

- **Process observed variables:**
  For each $u \in T$ assign relevant evidence to all $p_i \in \psi(u)$.
- **Compute the combined mini-functions:** Make an (i)-partitioning of $cluster_v(u)$, $\{mc(1), \ldots, mc(p)\}$;
- $h^1_{u \to v} = \sum_{elim(u,v)} \prod_{f \in mc(1)} f$
- $h^i_{u \to v} = max_{elim(u,v)} \prod_{f \in mc(i)} f \quad i = 2, \ldots, p$
- add $\{h^i_{u \to v} | i = 1, \ldots, p\}$ to $H_{u \to v}$. Send $H_{u \to v}$ to v.

2. **Compute upper bounds $\overline{P}(X_i, e)$ on $P(X_i, e)$:**
For every $X_i \in X$ let $u \in V$ be a cluster such that $X_i \in \chi(u)$. Make (i)-partition mini-clusters from $cluster(u)$, $\{mc(1), \ldots, mc(p)\}$; Compute $\overline{P}(X_i, e) =$
$(\sum_{\chi(u)-X_i} \prod_{f \in mc(1)} f) \cdot (\prod_{k=2}^{p} max_{\chi(u)-X_i} \prod_{f \in mc(k)} f)$.

Figure 8.15: Procedure Mini-Clustering for Belief Updating (MC-bel)



A
B
C
D
E
F
G
(a)
1 χ(1) = {A, B, C}
ψ(1) = {p(a), p(b|a), p(c|a,b)}
2 χ(2) = {B, C, D, F}
ψ(2) = {p(d|b), p(f|c,d)}
3 χ(3) = {B, E, F}
ψ(3) = {p(e|b,f)}
4 χ(4) = {E, F, G}
ψ(4) = {p(g|e,f)}
(b)
1 ABC
BC
2 BCDF
BF
3 BEF
EF
4 EFG
h(1,2)(b,c) = Σa p(a)·p(b|a)·p(c|a,b)
h(2,1)(b,c) = Σd,f p(d|b)·p(f|c,d)·h(3,2)(b,f)
h(2,3)(b,f) = Σc,d p(d|b)·p(f|c,d)·h(4,2)(b,c)
h(3,2)(b,f) = Σe p(e|b,f)·h(4,3)(e,f)
h(3,4)(e,f) = Σb p(e|b,f)·h(2,3)(b,f)
h(4,3)(e,f) = p(G = ge|e,f)
(c)

Figure 8.16: (a) A belief network; (b) A join-tree decomposition; (c) Execution of CTE-BU.

Algorithm *MC-bel* for upper bounds can be obtained from CTE by replacing step 2 of the main loop and the final part of computing the upper bounds on the joint belief by the procedure given in Figure 5.8 yielding the Algorithm in Figure 8.15. The partitioning of clusters to mini-clusters can be done in an identical manner to partitioning buckets into mini-buckets as discussed earlier.

Example 8.8.1 Figure 8.17 shows the trace of running MC-bel(3) on the problem in Figure 8.16. First, evidence $G = g_e$ is assigned in all CPTs. There are no individual functions to be sent from cluster 1 to cluster 2. Cluster 1 contains only 3 variables, $\chi(1) = \{A, B, C\}$, therefore it is not partitioned. The combined function $h^1_{1\to 2}(b,c) = \sum_a p(a) \cdot p(b|a) \cdot p(c|a,b)$ is computed and the message $H_{1\to 2} = \{h^1_{1\to 2}(b,c)\}$ is sent to node 2. Now, node 2 can send its message to node 3. Again, there are no individual functions. Cluster 2 contains 4 variables, $\chi(2) = \{B, C, D, F\}$, and a partitioning is necessary: MC-bel(3) can choose $mc(1) = \{p(d|b), h_{1\to 2}(b,c)\}$ and $mc(2) = \{p(f|c,d)\}$. The combined functions $h^1_{2\to 3}(b) = \sum_{c,d} p(d|b) \cdot h_{1\to 2}(b,c)$ and $h^2_{2\to 3}(f) = max_{c,d} p(f|c,d)$ are computed and the message $H_{4\to 3} = \{h^1_{2\to 3}(b), h^2_{2\to 3}(f)\}$ is sent to node 3. The algorithm continues until every node has received messages from all its neighbors. An upper bound on $P(a, G = g_e)$ can now be computed by choosing cluster 1, which contains variable A. It doesn't need partitioning, so the algorithm just computes $\sum_{b,c} p(a) \cdot p(b|a) \cdot p(c|a,b) \cdot h^1_{2\to 1}(b) \cdot h^2_{2\to 1}(c)$. Notice that unlike CTE-bel which processes 4 variables in cluster 2, MC-bel(3) never processes more than 3 variables at a time. □

It is easy to see that,

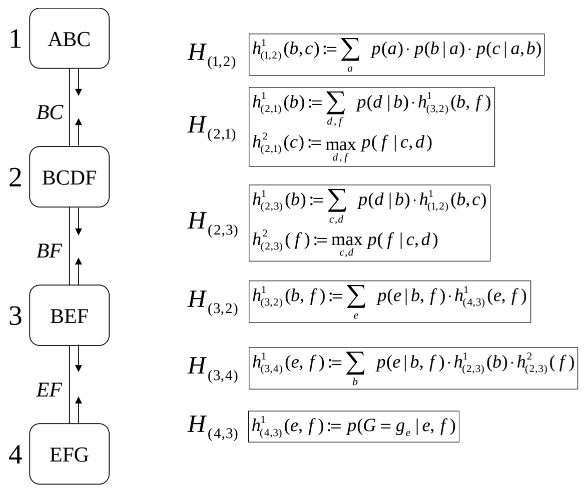

1 ABC
BC
2 BCDF
BF
3 BEF
EF
4 EFG
$H_{(1,2)}$ $h^1_{(1,2)}(b,c) := \sum_a p(a) \cdot p(b|a) \cdot p(c|a,b)$
$H_{(2,1)}$ $h^1_{(2,1)}(b) := \sum_{d,f} p(d|b) \cdot h^1_{(3,2)}(b,f)$
$h^2_{(2,1)}(c) := \max_{d,f} p(f|c,d)$
$H_{(2,3)}$ $h^1_{(2,3)}(b) := \sum_{c,d} p(d|b) \cdot h^1_{(1,2)}(b,c)$
$h^2_{(2,3)}(f) := \max_{c,d} p(f|c,d)$
$H_{(3,2)}$ $h^1_{(3,2)}(b,f) := \sum_e p(e|b,f) \cdot h^1_{(4,3)}(e,f)$
$H_{(3,4)}$ $h^1_{(3,4)}(e,f) := \sum_b p(e|b,f) \cdot h^1_{(2,3)}(b) \cdot h^2_{(2,3)}(f)$
$H_{(4,3)}$ $h^1_{(4,3)}(e,f) := p(G = g_e | e,f)$

Figure 8.17: Execution of MC-bel for $i = 3$

Theorem 8.8.2 *Given a Bayesian network $\mathcal{B} = \langle X, D, G, P \rangle$, MC-bel(i) computes an upper bound on the joint probability $P(X, e)$ of each variable and each of its values.*

You can also convince yourself about the complexity: (or consult the proof in [57])

Theorem 8.8.3 (Complexity of MC-bel(i)) *Given a Bayesian network $\mathcal{B} = \langle X, D, G, P \rangle$ and a tree-decomposition $\langle T, \chi, \psi \rangle$ of $\mathcal{B}$, the time and space complexity of MC-bel(i) is $O(n \cdot hw^* \cdot k^i)$, where n is the number of variables, k is the maximum domain size of a variable and $hw^* = max_{u \in T} |\{f \in P | scope(f) \cap \chi(u) \neq \Phi\}|$, which bounds the number of mini-clusters.*

Semantics of Mini-Clustering.

Mini-clustering generalizes the mini-bucket scheme which was shown to have the semantics of relaxation via *node duplication* [55, **?**] as follows. Given a tree-decomposition D, when computing a function $h_{(u,v)}$ (the message that cluster u sends to cluster v), cluster u is partitioned into p mini-clusters $u_1, ..., u_p$, which are first computed independently and then multiplied together. Instead consider a different decomposition D', which is just like D, with the exception that (a) instead of u, it has clusters $u_1, ..., u_p$, all of which are children of v, and each variable appearing in more than a single mini-cluster becomes a new variable, (b) each child w of u (in D) is a child of u_k (in D'), such that $h_{(w,u)}$ (in D) is assigned to

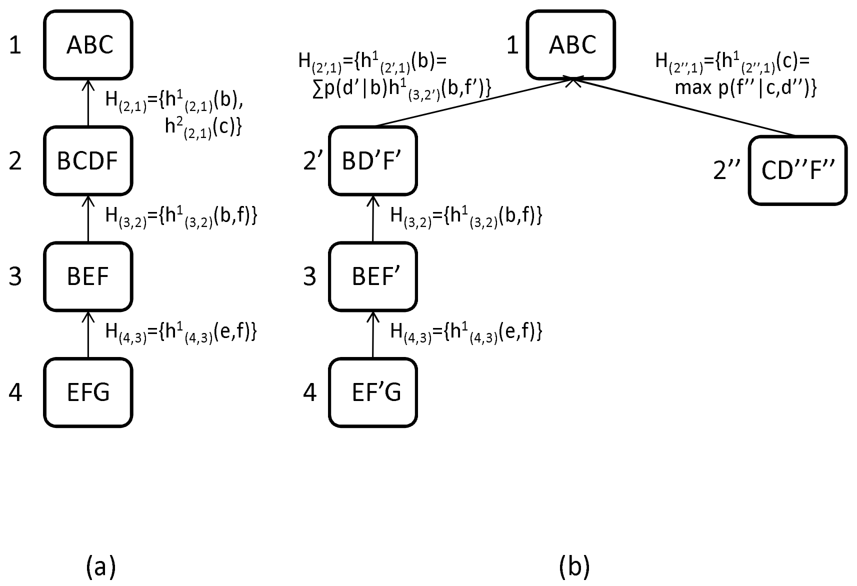


Figure 8.18: Node duplication semantics of MC: (a) trace of MC-bel(3); (b) trace of CTE-bel.

u_k (in $D^{'}$) during the partitioning. Note that $D^{'}$ is not a legal tree-decomposition relative to the original variables since it violates the connectedness property: the mini-clusters $u_1, ..., u_p$ contain variables $elim(u, v)$ but the path between the nodes $u_1, ..., u_p$ (this path goes through v) does not. However, it is a legal tree-decomposition relative to the new variables. It is straightforward to see that $H_{(u,v)}$ computed by MC-BU(i) on D is the same as $\{h_{(u_i,v)}|i = 1, ..., p\}$ computed by CTE-BU on $D^{'}$.

If we want to capture the semantics of the outward messages from root to leaves, we need to generate a different relaxed decomposition ($D^{''}$) because MC, as defined, allows a different partitioning in the up and down streams of the same cluster. We could of course stick with the decomposition in $D^{'}$ and use CTE in both directions which would lead to another variant of mini-clustering. (Exercise: design an algorithm that applied $CTE - bel$ to a Bayesian network that results from variable duplication and compare with the mini-clustering scheme).

Example 8.8.4 Figure 8.18(a) shows a trace of the bottom-up phase of MC-bel(3) on the network in Figure 8.17. Figure 8.18(b) shows a trace of the bottom-up phase of CTE-bel algorithm on a problem obtained from the problem in Figure 8.17 by duplicating nodes

D (into D' and D'') and F (into F' and F''). $\square$

The mini-clustering scheme for posterior marginals suffers from the same normalization issues as the mini-bucket scheme. As noted, MC-bel(i) is an improvement over the Mini-Bucket algorithm MBE-bel(i), in that it allows the computation of $\overline{P}(X_i, e)$ for all variables with a single run, whereas MBE(i) computes $\overline{P}(X_i, e)$ for just one variable, with a single run [82]. When computing $\overline{P}(X_i, e)$ for each variable, MBE-bel(i) has to be run n times, once for each variable (an algorithm we call nMBE(i)). In [55] it was demonstrated that MC-bel(i) has up to linear speed-up over nMBE(i). For a given i, the accuracy of MC-bel(i) can be shown to be not worse than that of nMBE(i).

Bibliography

[1] Darwiche A. *Modeling and Reasoning with Bayesian Networks.* Cambridge University Press, 2009.

[2] Bar-Yehuda R A. Becker and D. Geiger. Random algorithms for the loop-cutset problem. In *Uncertainty in AI (UAI'99)*, pages 81–89, 1999.

[3] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[4] S. A. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT*, 25:2–23, 1985.

[5] Reuven Bar-Yehuda, Dan Geiger, Joseph Naor, and Ron M. Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and bayesian inference. *SIAM J. Comput.*, 27(4):942–959, 1998.

[6] R. Bayardo and D. Miranker. A complexity analysis of space-bound learning algorithms for the constraint satisfaction problem. In *AAAI'96: Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 298–304, 1996.

[7] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *Uncertainty in AI (UAI'96)*, pages 81–89, 1996.

[8] Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the loop cutset problem. *J. Artif. Intell. Res. (JAIR)*, 12:219–234, 2000.

[9] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database ochemes. *Journal of the ACM*, 30(3):479–513, 1983.

[10] R.E. Bellman. *Dynamic Programming.* Princeton UNiversity Press, 1957.

[11] E. Bensana, M. Lemaitre, and G. Verfaillie. Earth observation satellite management. *Constraints*, 4(3):293–299, 1999.

[12] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming.* Academic Press, 1972.

[13] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming.* Academic Press, New York, 1972.

[14] B. Bidyuk and R. Dechter. On finding w-cutset in bayesian networks. In *Uncertainty in AI (UAI04)*, 2004.

[15] S. Bistarelli. *Semirings for Soft Constraint Solving and Programming (Lecture Notes in Computer Science.* Springer-Verlag, 2004.

[16] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the Association of Computing Machinery*, 44, No. 2:165–201, 1997.

[17] H.L. Bodlaender. Treewidth: Algorithmic techniques and results. In *MFCS-97*, pages 19–36, 1997.

[18] C. Cannings, E.A. Thompson, and H.H. Skolnick. Probability functions on complex pedigrees. *Advances in Applied Probability*, 10:26–61, 1978.

[19] Ming-Wei Chang, Lev-Arie Ratinov, and Dan Roth. Structured learning with constrained conditional models. *Machine Learning*, 88(3):399–431, 2012.

[20] A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 125(1-2):5–41, 2001.

[21] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association of Computing Machinery*, 7(3), 1960.

[22] S. de Givry, J. Larrosa, and T. Schiex. Solving max-sat as weighted csp. *In Principles and Practice of Constraint Programming (CP-2003)*, 2003.

[23] S. de Givry, I. Palhiere, Z. Vitezica, and T. Schiex. Mendelian error detection in complex pedigree using weighted constraint satisfaction techniques. In *ICLP Workshop on Constraint Based Methods for Bioinformatics*, 2005.

[24] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[25] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proc. Twelfth Conf. on Uncertainty in Artificial Intelligence*, pages 211–219, 1996.

[26] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference algorithms. In *Uncertainty in Artificial Intelligence (UAI'96)*, pages 211–219, 1996.

[27] R. Dechter. Bucket elimination: a unifying framework for processing hard and soft constraints. *CONSTRAINTS: An International Journal*, 2:51 – 55, 1997.

[28] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[29] R. Dechter. A new perspective on algorithms for optimizing policies under uncertainty. In *International Conference on Artificial Intelligence Planning Systems (AIPS-2000)*, pages 72–81, 2000.

[30] R. Dechter. *Constraint Processing.* Morgan Kaufmann Publishers, 2003.

[31] R. Dechter and Y. El Fattah. Topological parameters for time-space tradeoff. *Artificial Intelligence*, pages 93–188, 2001.

[32] R. Dechter and R. Mateescu. The impact of and/or search spaces on constraint satisfaction and counting. In *Proceeding of Constraint Programming (CP2004)*, pages 731–736, 2004.

[33] R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.

[34] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.

[35] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pages 353–366, 1989.

[36] R. Dechter and I. Rish. Directional resolution: The davis-putnam procedure, revisited. In *Principles of Knowledge Representation and Reasoning (KR-94)*, pages 134–145, 1994.

[37] R. Dechter and P. van Beek. Local and global relational consistency. *Theoretical Computer Science*, pages 283–308, 1997.

[38] Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.

[39] S. Even. Graph algorithms. In *Computer Science Press*, 1979.

[40] M. Fishelson, N. Dovgolevsky, and D. Geiger. Maximum likelihood haplotyping for general pedigrees. *Human Heredity*, 2005.

[41] M. Fishelson and D. Geiger. Exact genetic linkage computations for general pedigrees. *Bioinformatics*, 2002.

[42] Myan Fishelson and Dan Geiger. Optimizing exact genetic linkage computations. *RECOMB*, pages 114–121, 2003.

[43] Freuder. Partial constraint satisfaction. *Artificial Intelligence*, 50:510–530, 1992.

[44] E. C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

[45] M. R Garey and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness. In *W. H. Freeman and Company, San Francisco*, 1979.

[46] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[47] Nicola Leone Georg Gottlob and Francesco Scarcello. A comparison of structural csp decomposition methods. *Artificial Intelligence*, pages 243–282, 2000.

[48] V. Gogate. Sampling algorithms for probabilistic graphical models with determinism. Technical report, Ph.d. thesis, Information and Computer Science, Universiy of California, Irvine, 2009.

[49] Vibhav Gogate and Rina Dechter. Approximate inference algorithms for hybrid bayesian networks with discrete constraints. In *Proceeding of Uncertainty in Artificial Intelligence (UAI2005)*, 2005.

[50] Vibhav Gogate, Rina Dechter, Bozhena Bidyuk, Craig Rindt, and James Marca. Modeling transportation routines using hybrid dynamic mixed networks. In *UAI*, pages 217–224, 2005.

[51] H. Hasfsteinsson H.L. Bodlaender, J. R. Gilbert and T. Kloks. Approximating treewidth, pathwidth and minimum elimination tree-height. In *Technical report RUU-CS-91-1, Utrecht University*, 1991.

[52] R. A. Howard and J. E. Matheson. *Influence diagrams.* 1984.

[53] F.V. Jensen. *Bayesian networks and decision graphs.* Springer-Verlag, New-York, 2001.

[54] J. Larrosa K. Kask, R. Dechter and A. Dechter. Unifying tree-decompositions for reasoning in graphical models. *Artificial Intelligence*, 166(1-2):165–193, 2005.

[55] K. Kask and R. Dechter. A general scheme for automatic search heuristics from specification dependencies. *Artificial Intelligence*, pages 91–131, 2001.

[56] K. Kask, R. Dechter, J. Larrosa, and G. Fabio. Bucket-tree elimination for automated reasoning. *Submitted -2001*, 2001.

[57] R. Dechter K. Kask and J. Larrosa. A general scheme for multiple lower bound computation in constraint optimization. *Principles and Practice of Constraint Programming (CP2001)*, pages 346–360, 2001.

[58] U. Kjæaerulff. Triangulation of graph-based algorithms giving small total state space. In *Technical Report 90-09, Department of Mathematics and computer Science, University of Aalborg, Denmark*, 1990.

[59] D. Koller and N. Friedman. *Probabilistic Graphical Models.* MIT Press, 2009.

[60] J Larrosa and R. Dechter. Dynamic combination of search and variable-elimination in csp and max-csp. *Submitted*, 2001.

[61] J. Larrosa and R. Dechter. Boosting search with variable-elimination. *Constraints*, pages 407–419, 2002.

[62] Javier Larrosa and Rina Dechter. Boosting search with variable elimination in constraint optimization and constraint satisfaction problems. *Constraints*, 8(3):303–326, 2003.

[63] J.-L. Lassez and M. Mahler. On fourier's algorithm for linear constraints. *Journal of Automated Reasoning*, 9, 1992.

[64] S.L. Lauritzen and D.J. Spiegelhalter. Local computation with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50(2):157–224, 1988.

[65] D. Maier. The theory of relational databases. In *Computer Science Press, Rockville, MD*, 1983.

[66] Radu Marinescu and Rina Dechter. Memory intensive and/or search for combinatorial optimization in graphical models. *Artif. Intell.*, 173(16-17):1492–1524, 2009.

[67] R. Mateescu. And/or search spaces for graphical models. Technical report, Ph.d. thesis, Information and Computer Science, Universiy of California, Irvine, 2007.

[68] R. Mateescu and R. Dechter. The relationship between AND/OR search and variable elimination. In *Proceedings of the Twenty First Conference on Uncertainty in Artificial Intelligence (UAI'05)*, pages 380–387, 2005.

[69] R. Mateescu and R. Dechter. A comparison of time-space scheme for graphical models. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 2346–2352, 2007.

[70] Robert Mateescu and Rina Dechter. And/or cutset conditioning. In *International Joint Conference on Artificial Intelligence (Ijcai-2005)*, 2005.

[71] Robert Mateescu, Kalev Kask, Vibhav Gogate, and Rina Dechter. Join-graph propagation algorithms. *J. Artif. Intell. Res. (JAIR)*, 37:279–328, 2010.

[72] L. G. Mitten. Composition principles for the synthesis of optimal multistage processes. *Operations Research*, 12:610–619, 1964.

[73] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7(66):95–132, 1974.

[74] K. P. Murphy. *Machine Learning; a probabilistic perspective.* 2012.

[75] R.E. Neapolitan. *Learning Bayesian Networks.* Prentice hall series in Artificial Intelligence, 2000.

[76] N. J. Nillson. *Principles of Artificial Intelligence.* Tioga, Palo Alto, CA, 1980.

[77] P.Dagum and M.Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–155, 1993.

[78] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[79] L. Portinale and A. Bobbio. Bayesian networks for dependency analysis: an application to digital control. In *Proceedings of the 15th Conference on Uncertainty in Artifi cial Intelligence (UAI99)*, pages 551–558, 1999.

[80] A. Dechter R. Dechter and J. Pearl. Optimization in constraint networks. In *Influence Diagrams, Belief Nets and Decision Analysis*, pages 411–425. John Wiley & Sons, 1990.

[81] E. Bin R. Emek R. Dechter, K. Kask. Generating random solutions for constraint satisfaction problems. In *Proceedings of the National Conference of Artificial Intelligence (AAAI-02)*, 2002.

[82] R Dechter R. Mateescu and K. Kask. Tree approximation for belief updating. In *National Conference of Artificial Intelligence (AAAI-2002)*, pages 553–559, 2002.

[83] B. D'Ambrosio R.D. Shachter and B.A. Del Favero. Symbolic probabilistic inference in belief networks. In *National Conference on Artificial Intelligence (AAAI'90)*, pages 126–131, 1990.

[84] I. Rish and R. Dechter. Resolution vs. search; two strategies for sat. *Journal of Automated Reasoning*, 24(1/2):225–275, 2000.

[85] Irina Rish and Rina Dechter. Resolution versus search: Two strategies for sat. *J. Autom. Reasoning*, 24(1/2):225–275, 2000.

[86] Emma Rollon and Rina Dechter. New mini-bucket partitioning heuristics for bounding the probability of evidence. In *AAAI*, 2010.

[87] D. Roth. On the hardness of approximate reasoning. 82(1-2):273–302, April 1996.

[88] D. G. Corneil S. A. Arnborg and A. Proskourowski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Discrete Mathematics.*, 8:277–284, 1987.

[89] T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. *Proc. IJCAI-99*, pages 542–547, 1999.

[90] L. K. Saul and M. I. Jordan. Learning in boltzmann trees. *Neural Computation*, 6:1173–1183, 1994.

[91] R. Seidel. A new method for solving constraint satisfaction problems. In *International Joint Conference on Artificial Intelligece (Ijcai-81)*, pages 338–342, 1981.

[92] G. R. Shafer and P.P. Shenoy. Axioms for probability and belief-function propagation. volume 4, 1990.

[93] P.P. Shenoy. Valuation-based systems for bayesian decision analysis. *Operations Research*, 40:463–484, 1992.

[94] P.P. Shenoy. Binary join trees for computing marginals in the shenoy-shafer architecture. *International Journal of approximate reasoning*, pages 239–263, 1997.

[95] K. Shoiket and D. Geiger. A proctical algorithm for finding optimal triangulations. In *Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 185–190, 1997.

[96] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM Journal of Computation.*, 13(3):566–579, 1984.

[97] P. Thbault, S. de Givry, T. Schiex, and C. Gaspin. Combining constraint processing and pattern matching to describe and locate structured motifs in genomic sequences. In *Fifth IJCAI-05 Workshop on Modelling and Solving Problems with Constraints*, 2005.

[98] Bozhena Bidyuk Craig Rindt Vibhav Gogate, Rina Dechter and James Marca. Modeling transportation routines using hybrid dynamic mixed networks. In *Proceeding of Uncertainty in Artificial Intelligence (UAI2005)*, 2005.

[99] Yair Weiss and Judea Pearl. Belief propagation: technical perspective. *Commun. ACM*, 53(10):94, 2010.

[100] N.L. Zhang and D. Poole. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research (JAIR)*, 1996.