CHAPTER 8

Bounding Inference: Decomposition Bounds

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, especially 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, a search scheme or their hybrid. Bounded inference algorithms approximate inference schemes, while sampling schemes can be viewed as approximating search. The word "bounded" refers to our desire to bound the size of functions created during inference. Yet, it also often comes with algorithms that provide upper and lower bounds on the results.

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 that are also referred to as decomposition bounds, such as the mini-bucket, mini-clustering and iterative join-graph propagation, often accompanied with re-parameterizations, or cost-shifting. These methods offer an adjustable trade-off between accuracy and efficiency within some memory resource.

It was shown that deriving general approximation scheme with a guaranteed relative error bounds, is NP-hard [32, 114]. One alternative for dealing with these bleak fact is to develop *anytime algorithms*. Such algorithms produce better and better solutions accompanied with tighter bounds and can therefore be responsive to the users allowing to be interrupted at any time while producing the best solution found thus far.

As we showed (Chapter **??**), the bucket-elimination scheme is a unifying algorithmic scheme for variable-elimination algorithms that is widely applicable. These include *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 algorithms for mixed optimization and summation such as map (or marginal map) and influence diagrams [48].

Decomposition bounds are not fully anytime because they may require significant memory on their way to exact solution. Yet they can facilitate anytime schemes if they are used as heuristic functions within an anytime search algorithm [92, 93].

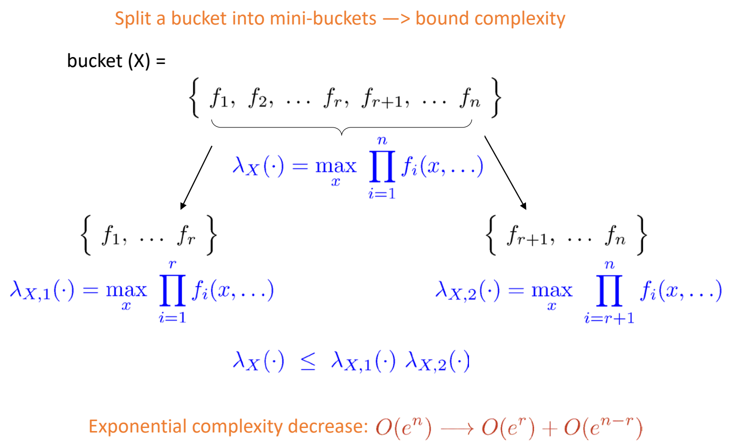

Split a bucket into mini-buckets —> bound complexity
bucket (X) =
$\{ f_1, f_2, \dots f_r, f_{r+1}, \dots f_n \}$
$\lambda_X(\cdot) = \max_x \prod_{i=1}^{n} f_i(x,\dots)$
$\{ f_1, \dots f_r \}$
$\{ f_{r+1}, \dots f_n \}$
$\lambda_{X,1}(\cdot) = \max_x \prod_{i=1}^{r} f_i(x,\dots)$
$\lambda_{X,2}(\cdot) = \max_x \prod_{i=r+1}^{n} f_i(x,\dots)$
$\lambda_X(\cdot) \le \lambda_{X,1}(\cdot)\,\lambda_{X,2}(\cdot)$
Exponential complexity decrease: $O(e^n) \longrightarrow O(e^r) + O(e^{n-r})$

Figure 8.1: The idea of mini-bucket approximation.

8.1 MINI-BUCKET APPROXIMATION FOR MPE

Consider the bucket-elimination algorithm *BE-mpe* (Chapter **??**). 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 $h_1, \dots, h_t$ be the functions residing in the bucket of X_p, Ψ_p is the input function and let $S_1, \dots, S_t$ be their respective scopes. Recall that 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 exchanging summation and multiplication. Since, in general, for any two non-negative functions $Z(x)$ and $Y(x)$, $\max_x Z(x) \cdot Y(x) \le \max_x Z(x) \cdot \max_x Y(x)$, this approximation will yield an upper bound on h_p. In particular, the function $g_p = \max_{X_p} \psi_P \cdot \prod_{i=1}^t \max_{X_p} h_i$, 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, thus requiring less computation.

The idea is illustrated in Figure 1.1, 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 $r \le n$. In general, the functions $h_1, \dots, h_t$ can be partitioned into any set of subsets called *mini-buckets*. Let $Q = \{Q_1, \dots, Q_r\}$ be a partitioning into mini-buckets of the functions $h_1, \dots, h_t$ in X_p's bucket (these include both the original functions or messages sent from other buckets). Assume that the mini-bucket Q_l contains the functions $h_{l_1}, \dots, h_{l_r}$. 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, instead: $g_p = \prod_{l=1}^{r} \max_{X_p} \prod_{h \in Q_l} h$. Each new mini-bucket function (or message), $\max_{X_p} \prod_{h \in Q_l} h$, is computed independently and is 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 exact messages with their upper bounds, it is easy to see that once all buckets are processed, the maximized product computed in the first bucket is an upper bound on the MPE value. A lower bound can now 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 whose cost will serve as a lower-bound, since it is not necessarily optimal.

It is convenient to control the algorithm's performance using two bounding parameters. Parameter called i-bound whose value is 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 1.2.

Clearly, for efficiency, we would want the mini-buckets to be as small as possible yet we wish the scheme to be as accurate as possible. In general, as m and i increase, we get tighter bounds. Note however, that a monotonic increase in accuracy as a function of the i-bound i, can be guaranteed only under the refinement condition discussed next.

Definition 8.1 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 8.2 *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'', or $Q'_j = \{Q''_{j_1}, ..., Q''_{j_l}\}$. Therefore,

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

□

Definition 8.3 (i,m)-partitioning. A partitioning of $h_1, ..., h_t$ is *canonical* if any function f whose scope is subsumed by another's, is placed into a bucket containing one of those subsuming

```
Algorithm MBE-mpe(i,m)
Input: A belief network $\mathcal{B} =< X, D, G, \mathcal{P}_G, \prod >$, an ordering of the variables, $d = X_1, ..., X_n$; evidence 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 a 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.
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:
    • Compute an mpe cost by maximizing over $X_1$, the product in $bucket_1$. Namely $U \leftarrow max_{X_1} \prod_{h_j \in bucket_1} h_j$.
    • (Generate an approximate mpe tuple): For $i = 1$ to $n$ along $d$ do: Given $\mathbf{x}_{(1...(i-1))} = (x_1, \ldots, x_{i-1})$ choose $x_i = argmax_{X_i} \prod_{\{h_j \in \ bucket_i\}} h_j(\mathbf{x}_{(1...(i-1))})$.
      $L \leftarrow P(x_1, ..., x_n)$
4. Output: $U$ and $L$ and configuration: $\bar{x} = (x_1, ..., x_n)$
```

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

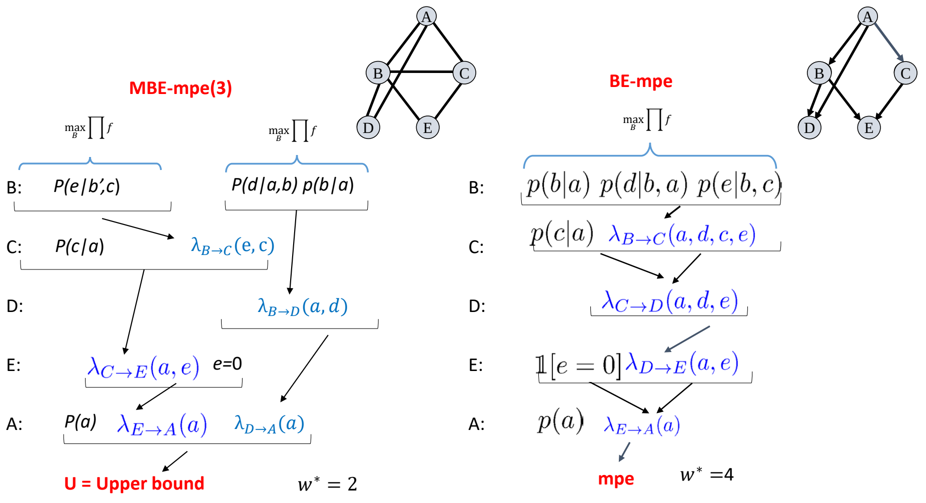

MBE-mpe(3)
$\max_B \prod f$
$\max_B \prod f$
B: P(e|b',c)
P(d|a,b) p(b|a)
C: P(c|a)
$\lambda_{B\to C}(e,c)$
D: $\lambda_{B\to D}(a,d)$
E: $\lambda_{C\to E}(a,e)$ e=0
A: P(a) $\lambda_{E\to A}(a)$ $\lambda_{D\to A}(a)$
U = Upper bound
$w^* = 2$
BE-mpe
$\max_B \prod f$
B: $p(b|a)\ p(d|b,a)\ p(e|b,c)$
C: $p(c|a)\ \lambda_{B\to C}(a,d,c,e)$
D: $\lambda_{C\to D}(a,d,e)$
E: $\mathbb{1}[e=0]\lambda_{D\to E}(a,e)$
A: $p(a)$ $\lambda_{E\to A}(a)$
mpe
$w^* = 4$
A
B
C
D
E

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

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 i-bound, i (number of variables) and the m-bound, m (number of functions), are not independent in the sense that 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 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. For flexibility reasons the i-bound would be the one mostly used in practice.

Clearly, since *MBE-mpe(i,m)* computes an upper bound in each bucket it yields an overall upper bound on the resulting mpe. Also, a lower bound is obtained as the cost of the suboptimal solution generated.

Theorem 8.4 MBE-mpe boundness. *Algorithm* MBE-mpe*(i, m) computes an upper and a lower bounds on the mpe.*

Example 8.5 Figure 1.3 compares algorithms *BE-mpe* and *MBE-mpe(i,m)* where $i = 3$ and $m = 2$ over the network given in the figure (the directed network and its moral graph) along the ordering $o = (A,\ E, D,\ C, B)$. Notice that in this figure we use the bucket-tree elimination notation where messages are indexed by their origin and destination bucket. The exact *BE-mpe* sequentially records the new functions (shown to the right) $\lambda_{B\to C}(a, d, c, e)$, $\lambda_{C\to D}(a, d, e)$, $\lambda_{D\to E}(a, e)$, and $\lambda_{E\to A}(a)$. Then, in the bucket of A, it computes $M = \max_a P(a)\lambda_{E\to A}(a)$. Subsequently, an mpe configuration $(A = a', B = b', C = c', D = d', E = e')$ where $e' = 0$ is the evidence, can be computed along d by selecting a value that maximizes the product of functions in the corresponding buckets conditioned on the previously assigned values. Namely, $a' = \arg\max_a P(a)\lambda_{E\to A}(a)$, $e' = 0$, $d' = \arg\max_d \lambda_{C\to D}(a, d, e)$, 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 the left handside of Figure 1.3. There can be several (3,2)-partitionings, and any choice would be legitimate, and can be selected arbitrarily. The new functions $\lambda_{B\to C}(e, c)$ and $\lambda_{B\to D}(d, a)$ 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 variable A the product of functions in A's bucket: $U = max_a P(a)\lambda_{E\to A}(a)\lambda_{D\to A}(a)$.

Once all the buckets are processed, a suboptimal mpe solution can be computed by instantiating each variable by one of its values that maximizes the product of functions in the corresponding bucket, in the same way this is done by exact *BE-mpe*. The probability of any assignment to all the variables is a lower bound on the mpe, and one can use alternative methods to compute a full (sub-optimal) solution (i.e., a configuration) given the functions recorded by the mini-bucket algorithm. Note that by design *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

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

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.

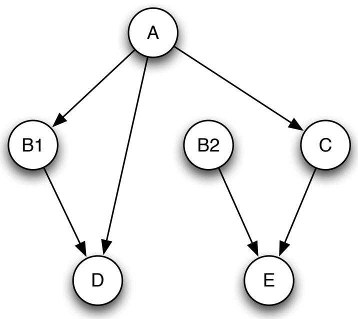
8.1.1 THE MINI-BUCKET SEMANTICS

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 associated with a single mini-bucket. For example, the Mini-Bucket trace in Figure 1.3b, corresponds to solving exactly by full bucket-elimination the network of the problem in Figure 1.4. 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 happens to be 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 the duplicated variables, we know that the assignment is locally optimal, namely conditioned on the current partial assignment.

8.2 MINI-BUCKET APPROXIMATION FOR BELIEF UPDATING

As shown in Chapter **??**, 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* computes $P(x_1, \mathbf{e})$, when X_1 is the first variable and then computes $P(x_1|\mathbf{e}) = \alpha P(x_1, \mathbf{e})$ where α is the normalization constant (see Chapter 4).



(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 1.3b

```
Algorithm mbe-bel-max(i,m)
Input: A belief network $\mathcal{B} =< X, D, G, \mathcal{P}_G, \prod >$, an ordering of the variables, $d = X_1, ..., X_n$; evidence e.
Output: an upper bound on $P(x_1, \bar{e})$ and an upper bound on $P(e)$.
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.
2. Backward: For $p \leftarrow n$ downto 1, do
for all the functions $h_1, h_2, ..., h_j$ in $bucket_p$, do
• If $X_p$ is observed $(X_k = x_k)$, assign $X_k \leftarrow x_p$ in each $h_j$ and put the result
  in the highest-variable bucket of its scope (put constants in $bucket_1$).
  • Else
    Generate an $(i, m)$-partitioning, $Q' = \{Q_1, ..., Q_r\}$ of $h_1, ..., h_t$ in $bucket_p$.
    For each $Q_l \in Q'$, containing $h_{l_1}, ...h_{l_t}$, do
        If $l = 1$ compute $h^l \leftarrow \sum_{X_k} \prod_{j=1}^{t} h_{1_j}$
        Else compute $h^l \leftarrow max_{X_k} \prod_{j=1}^{t} h_{l_j}$
        Add $h^l$ to the bucket of the highest-index variable in scope($h^l$)
        (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 $P(x_1, \bar{e})$.
$P'(\bar{e}) \leftarrow \sum_{x_1} P'(x_1, \bar{e})$, which is an upper bound on probability of evidence.
```

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

The mini-bucket idea can be applied now 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 apply summation in each mini-bucket, we will get the approximate bound of h_p by $f_p = \prod_{l=1}^{r} \sum_{X_p} \prod_{l_i} h_{l_i}$. 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 $\sum_{X_p} \prod_{l_i} h_{l_i}$, a constant function relative to X_p. Instead, let's distinguish one (arbitrary) mini-bucket (the first one here) and 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_p by its sum over X_p, we can bound $(i > 1)$, by its maximum operator over X_p, yielding

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

Therefore, an upper bound g_p of h_p is obtained by summing over one of X_p's mini-buckets and maximizing over the rest. This will lead to a tighter bound.

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 1.5. Algorithms *MBE-bel-min* and *MBE-bel-mean* can be obtained by replacing the operator max by min and by $mean$, respectively. The *mean* operator is like the summation operator, divided by the number of elements in the sum. Notice however that the duplication semantics of MBE which implies summation for each of the mini-buckets does not extend here. This assymetry in mini-bucket processing for summation queries will be generalized and improved shortly using the notion of weighted mini-buckets. We can show that

Theorem 8.6 *Given a Bayesian network with evidence* $\mathbf{e}$*, algorithm MBE-bel-max(i,m) computes an upper bound on* $P(X_1, \mathbf{e})$ *and on* $P(\mathbf{e})$*, respectively.*

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 which mini-bucket should be operated by summation and which by maximization.

Proposition 8.7 *The following holds:*

1. *For every partitioning Q of a set of functions whose scopes include variable* X_p*,* $h_p \leq g_Q^p \leq f_Q^p$*, 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 $Q^{'}$, whose sumed mini-bucket is a refinement of the sumed mini-bucket in $Q^{'}$, 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 not 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 $P(X_1, \mathbf{e})$ 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.

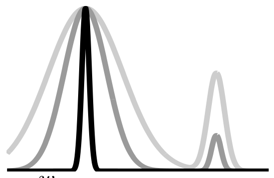
8.3 WEIGHTED MINI-BUCKET ELIMINATION

The asymmetry in processing mini-buckets for summation queries is annoying, especially when we lack any criterion for choosing the summation mini-bucket. We will next see a generalization of the mini-bucket scheme into *weighted mini-bucket (wmb)* which leads to tighter bounds by associating mini-buckets with weights used in Holder inequality [19]. We first define the notion of a power-sum.

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

where w is a non-negative weight. The power sum reduces to a standard summation when $w = 1$ and approaches max when $w \to 0^+$ illustrated by Figure 1.6. For the case of 2 functions, when $w = w_1 + w_2$, Holder inequality looks as follows.

$$\sum_x^w f_1(x) \cdot f_2(x) \leq (\sum_x f_1(x)^{\frac{1}{w_1}})^{w_1} \cdot (\sum_x f_2(x)^{\frac{1}{w_2}})^{w_2}$$



$$\lim_{w \to 0^+} \sum_x^w f(x) = \max_x f(x)$$

Figure 8.6: The impact of w on the power sum.

The general Holder inequality is:

Proposition 8.8 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 the summation operator to the power-sum for 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. (1.2). 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, any choice of mini-bucket weights will tighten the obtained bound. It is easy to see that uniform weights will yield superior bounds to simple summation (that correspond to node duplication) (prove as an exercise). 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, corresponds to assigning one mini-bucket with $w = 1$ and the rest with $w = 0$.

Algorithm weighted mini-bucket (WMB) is given in Figure 1.7. It is parameterized 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 the weights based on variables, so that buckets of sum variables will be assigned $w = 1$ and others $w = 0$ for maximization. This makes WMB algorithm not only poetntially better, by optimizing the weight parameters, but also provide a uniform way for expressing the mini-bucket scheme.

Weighted mini-buckets for lower bounds. WMB can also be used to compute a lower bound when the factors are strictly positive; this is obtained by requesting that exactly one weight, say, w_{l_k} is positive, and the rest are negative: $w_{l_k} > 0, w_{l_j} < 0$ [19].

8.4 MINI-BUCKET ELIMINATION FOR MARGINAL 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 $A = \{A_1, ..., A_k\}$, which are the MAP variables, and evidence $\mathbf{e}$, the problem is to find an assignment to the hypothesized variables that maximizes their

```
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 $P(x_1, e)$
1. Initialize: Partition $P = \{P_1, ..., P_n\}$ into buckets $bucket_1, \ldots, bucket_n$, 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
          $$h_l \leftarrow \sum_{X_k}^{w_l} \prod_{j=1}^{t} h_{l_j} = (\sum_{X_k} \prod_{j=1}^{t} (h_{l_j})^{\frac{1}{w_l}})^{w_l}$$
      Add $h_l$ to the bucket of the highest-index variable in its scope.
3. Return (as a result of processing $bucket_1$):
1. The product of dunction is $bucket_1$ which is an upper-bound on $P(X_1, e)$.
2. $U_1 \leftarrow$ the sum over $X_1$ of the product of functions in the bucket of $X_1$, which is an upper bound on probability of evidence.
```

Figure 8.7: Algorithm *WMBE(i,m)*.

```
Algorithm WMBE-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 $A = a$.
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 $bucket_p$: $w_{p_1}, ..., w_{p_r}$. s.t $\sum_i w_{p_i} = w_p$.
      foreach $Q_l \in Q'$, containing $h_{l_1}, ...h_{l_t}$, do
          $$h_l \leftarrow \sum_{X_k}^{w_{p_l}} \prod_{j=1}^{t} h_{l_j} = (\sum_{X_k} (\prod_{j=1}^{t} h_{l_j})^{\frac{1}{w_{p_l}}})^{w_{p_l}}$$
      Add $h_l$ to the bucket of the highest-index variable in its scope.
3. Forward: for $p = 1$ to $k$, given $A_1 = a_1^a, ..., A_{p-1} = a_{p-1}^a$, assign a value $a_p^a$ 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, computed in the first bucket, and the assignment $a = (a_1^a, ..., a_k^a)$.
```

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

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 MAP and SUM variables, respectively. Since $P(\mathbf{e})$ is a normalization constant, the same maximum is obtained for $P(\bar{a}_k, \mathbf{e})$.

As we know, the bucket-elimination algorithm for finding the exact mmap, *BE-map* 1.5, assumes constrained orderings in which the MAP variables appear first and thus are processed last by bucket-elimination 1.5. This restriction makes this MMAP task more difficult because it implies higher induced widths. The algorithm has the usual backward phase while its forward phase is relative to the MAP 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 as a weighted mini-bucket scheme. As usual, we partition each bucket into mini-buckets. If the bucket's variable can be eliminated by summation, we apply the power-sum with $w = 1$, where the selected mini-bucket weights is left as a hyper-parameter. The rest of the buckets are processed by the mini-bucket rule with $w = 0$, yielding maximization. In other words, when the algorithm reaches the MAP buckets, their processing is identical to that of *MBE-mpe*. Algorithm *WMBE-map(i,m)* is described in Figure 1.8.

Decoding the map assignment and the issue with lower-bounds. Once the backwards phase of *MBE-map* ends, we have an upper bound and, in principle, we can compute a map assignment in the forward phase. While the probability of any assignment is a lower bound on the MMAP 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 *WMBE* in the buckets of summation variables since those served as upper bounds. One possibility is, given an assignment to the MAP variables, is to rerun the mini-bucket algorithm over the SUM variables using the min operator with $w = -1$, conditional on the map assignment and subsequently compute a lower bound on the assigned tuple in another forward step over the first k buckets. We leave the details of this idea as an exercise.

Example 8.9 Consider a belief network which describes the decoding of a *linear block code*, shown in Figure 1.9. 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):
$bucket(X_0) = P(y_0^x|X_0), P(X_0|U_0, U_1, U_2),$
$bucket(X_1) = P(y_1^x|X_1), P(X_1|U_1, U_2, U_3),$
$bucket(X_2) = P(y_2^x|X_2), P(X_2|U_2, U_3, U_4),$
$bucket(X_3) = P(y_3^x|X_3), P(X_3|U_3, U_4, U_0),$
$bucket(X_4) = P(y_4^x|X_4), P(X_4|U_4, U_0, U_1),$
$bucket(U_0) = P(U_0), P(y_0^u|U_0),$
$bucket(U_1) = P(U_1), P(y_1^u|U_1),$
$bucket(U_2) = P(U_2), P(y_2^u|U_2),$

$bucket(U_3) = P(U_3), P(y_3^u|U_3),$
$bucket(U_4) = P(U_4), P(y_4^u|U_4).$

Processing by *MBE-map(4,1)* 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 messages. Three are placed in bucket U_0 ($h_{X_0}, h_{X_3}, h_{X_4}$), one in bucket U_1 (h_{U_0}) and one in bucket U_2 (h_{X_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 $Upper$ on the map value, namely $Upper \geq P(U, \bar{y}^u, \bar{y}^x)$ 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 many 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 $P(U_1, \mathbf{e})/P(U_2, \mathbf{e})$.

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

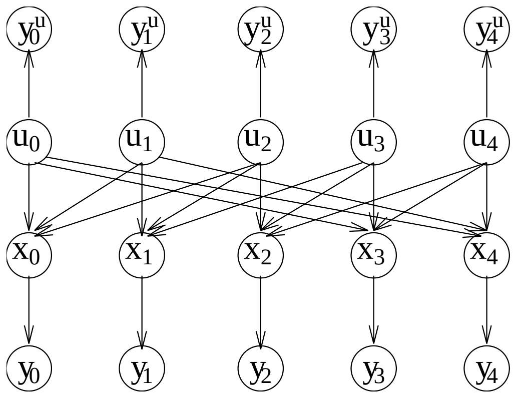


Figure 8.9: Belief network for a linear block code.

```
Algorithm MBE-opt(i,m)
Input: A cost network $(X, D, F, \sum)$, $F = \{C_1, ..., C_l\}$; ordering $d$.
Output: A lower and an upper bound on the optimal cost.
1.Initialize: Partition $C$ into $bucket_1, \ldots, bucket_n$, 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 $(X_p = x_p)$, replace $X_p$ by $x_p$ 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 $X_1 = x_1, ..., X_{p-1} = x_{p-1}$,
  assign a value $x_p$ to $X_p$ that minimizes the sum of all functions in $bucket_p$.
4. Output the assignment $x = (x_1, ..., x_n)$, an upper bound $U = F(x)$, and a lower bound on the
  optimal cost $L = min_{X_1} \sum_{h_i \in bucket_1} h_i$.
```

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

optimization [16]. 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 $C = \sum_{i=1}^{l} C_i$. 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 in Figure 1.10. 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 provides another one way to generalize this scheme, leading to Weighted mini-bucket elimination [87].

8.6 COMPLEXITY AND TRACTABILITY

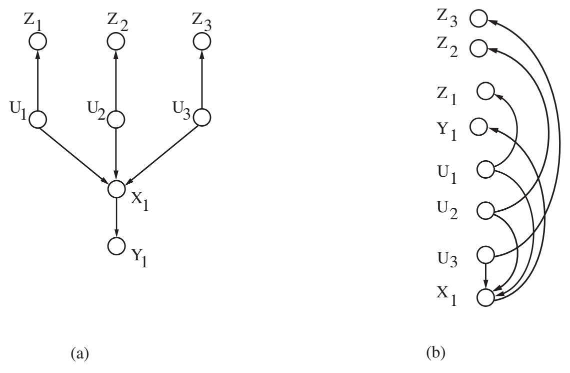
8.6.1 THE CASE OF LOW INDUCED WIDTH

We denote by *weighted mini-bucket-elimination(i,m)*, or simply *WMB(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 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 obeys the following:

Theorem 8.10 *Algorithm* WMB(i,m) *takes* $O(r \cdot k^i)$ *time and space, where,* k *bounds the domain size and* r *is the number of input functions*[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,* $|S| \leq i \leq n$.

Proof. 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, denoted $ch(v)$ where children correspond to all the functions in the corresponding mini-bucket). 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 where i bounds the resulting scope size of the 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 Bayesian networks. □

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



(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.

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)*,if applied along a proper orderings, is exact for acyclic networks and in particular for polytrees. 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 **??**). 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. This is illustrated in Example 1.11 as elaborated next.

Example 8.11 Consider an ordering $d = (X_1, U_3, U_2, U_1, Y_1, Z_1, Z_2, Z_3)$ of the polytree in Figure 1.11a, where the last four variables $Y_1,\ Z_1,\ Z_2,\ Z_3$ in the ordering are observed. Once those four buckets are processed as observation buckets, we get (where observed values shown in low-case):
$bucket(U_1) = P(U_1), P(X_1|U_1, U_2, U_3), P(z_1|U_1),$
$bucket(U_2) = P(U_2), P(z_2|U_2),$
$bucket(U_3) = P(U_3), P(z_3|U_3)$
$bucket(X_1) = P(y_1|X_1).$
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.

Theorem 8.12 *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.*

8.6.2 HEURISTIC FOR MINI-BUCKET PARTITIONINGS

Clearly, given the parameters i and m, there are many (i, m) partitioning. These partitionings can be guided by the graph, or by the content of the actual functions, aiming to minimize the error incurred by the partitioning. 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 contains regardless of the number of function it has because controlling the mini-bucket size by the number of variables allows more flexibility. 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)*.

Scope-based Partitioning Heuristic. The *scope-based* partition heuristic (SCP) aims at minimizing the number of mini-buckets in the partition by including in each mini-bucket as many functions as possible as long as the i bound is satisfied. First, single function mini-buckets are decreasingly ordered according to their arity. Then, each mini-bucket, from first to last, is absorbed into the earliest mini-bucket with whom it can be merged. See 1.12. The time and space complexity of `Partition`(B, i) , where B is the partitioned bucket, using the SCP heuristic is $O(|B| \log(|B|) + |B|^2)$ where $|B|$ is the number of variables in the bucket. (Exercise: prove this complexity).

Content-based Partitioning Heuristic. 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 $Q^* = \arg\min_Q dist(g_Q, g)$. Distance measures that can be considered include *log relative error*, *maximum log relative error*, *KL divergence* and *absolute error*. For example:

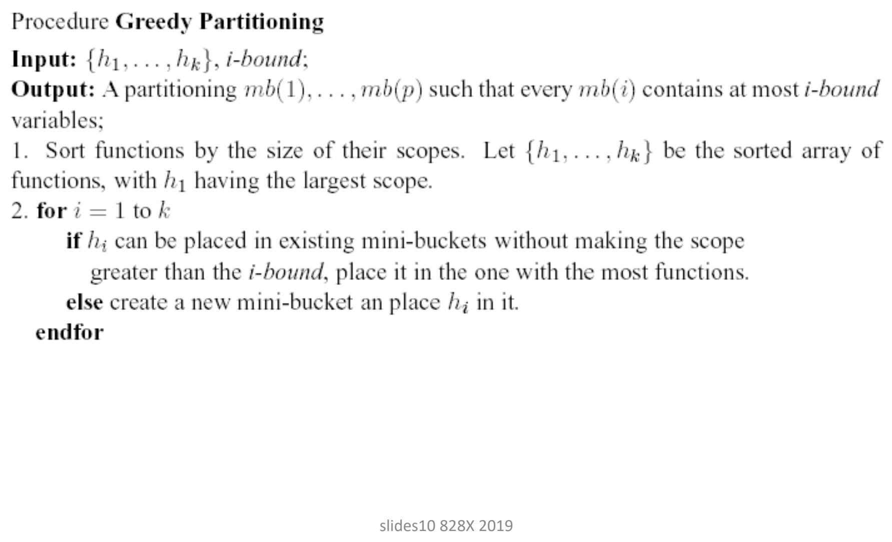
- *Log relative error, (RE)*:

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

- *Max log relative error (MRE)* :

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

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



```
Procedure **Greedy Partitioning**
**Input:** $\{h_1, \ldots, h_k\}$, *i-bound*;
**Output:** A partitioning $mb(1), \ldots, mb(p)$ such that every $mb(i)$ contains at most *i-bound*
variables;
1. Sort functions by the size of their scopes. Let $\{h_1, \ldots, h_k\}$ be the sorted array of
functions, with $h_1$ having the largest scope.
2. **for** $i = 1$ to $k$
      **if** $h_i$ can be placed in existing mini-buckets without making the scope
        greater than the *i-bound*, place it in the one with the most functions.
      **else** create a new mini-bucket an place $h_i$ in it.
   **endfor**
```

slides10 828X 2019

Figure 8.12: Algorithm SCP

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, as dictated by Proposition 1.2 .

Example 8.13 Consider a bucket $\mathcal{B}_x = \{f_1, f_2, f_3, f_4\}$. Its Hasse diagram is depicted in Figure 1.13. As observed, the finest partition is $Q^{\perp} = \{\{f_1\}, \{f_2\}, \{f_3\}, \{f_4\}\}$ (depicted 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).

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 were considered. A guiding heuristic function along the lattice can be constructed based on the error that occurs 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 [21]). At each step, the algorithm ranks

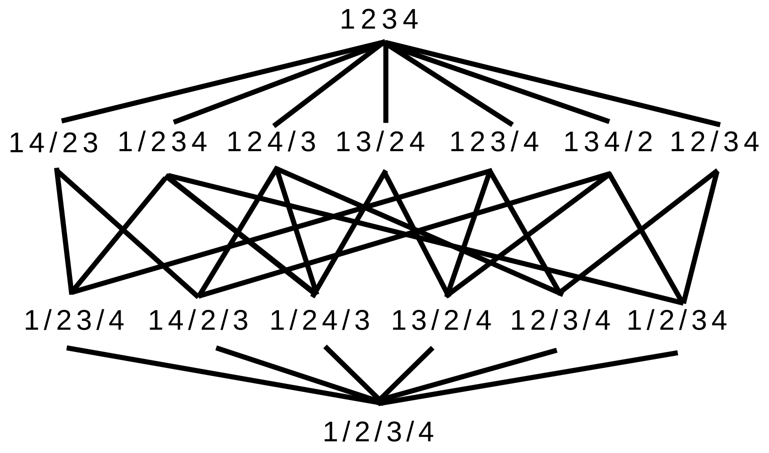


Figure 8.13: Partitioning lattice of bucket $\{f_1, f_2, f_3, f_4\}$. We specify each function by its subindex.

```
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 8.14: Greedy partitioning

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 1.14.

Proposition 8.14 *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.*

8.6.3 DISCUSSION

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 just a 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 variable C and then go back and process another mini-bucket of variable B and so on. This alternative schedule is apparent once we reason about partitioning heuristics as variable duplications. Finally, a relaxed network due to

```
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, $d = X_1, ..., X_n$; evidence $\mathbf{e}$.
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 8.15: Algorithm *ANYTIME-mpe(ϵ)*.

variable-duplication can be processed by any means, not necessarily by bucket-elimination. This observation opens up a richer collection of bounding schemes that can be considered.

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. An important property 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 up to a point. *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, evalntually generating the optimal solutionis memory allows.

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 (i.e., memory) are exhausted. To illustrate, such an anytime scheme for finding the mpe, *ANYTIME-mpe(ϵ)* is presented in Figure 1.15, where ϵ is a 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 (i.e., memory) are exhausted, returning the largest lower bound and the smallest upper bound found so far, as well as the current highest suboptimal solution. This scheme is not fully anytime due to the memory restriction of MBE. In other words, it is not *any space*.

For summation queries such as belief computations, deriving the upper and lower bound can be done using two runs of WMBE to generate both an upper bound and a lower bound using all-positive, or a single negative weight, respectively.

Mini-bucket as heuristic generation for search. 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 for maximization), as it is done in the forward step of *MBE-mpe*, we can continue searching for better solutions (e.g., by branch and bound schemes), while using the mini-bucket functions to guide and prune the search. This approach was explored extensively in recent years yielding state-of the art algorithms both for finding an mpe assignment as well as for constraint optimization problems [7, 93].

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 [69**? ? ?**].

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. Tus, we would not need to run the algorithm for every variable independently. We focus on likelihood computations (belief-updating and probability of evidence) for which such extensions (from mini-bucket to mini-clustering) are most relevant. 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 **??**). During CTE's processing of a cluster u, the cluster contains the original functions as well as messages from neighboring clusters which we denote as $cluster(u)$ (see Algorithm **??**). 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 from node u to node v; we can divide the functions of $cluster(u)$ into p mini-clusters and rewrite $h_{u\to v} = \sum_{elim(u,v)} \prod_{k=1}^{p} \prod_{f\in mc(k)} f$. 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 will be described using the weight notation and is called *Weighted Mini-Clustering (WMB(i)* is presented in Figure 1.16 (notice that we dropped the m parameter for simplicity).

The messages which is a set of functions is referred to as a *combined functions*. The combined function from u to v is computed 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 and each associated with a weight w_j s.t. $\sum_j w_j = 1$. Each mini-cluster is processed by the power-sum over the eliminator and the resulting set of functions as well as all the individual functions (that are 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 associating one mini-cluster with a negative weight (e.g., replacing the max operator with min operator).

Algorithm *WMC-bel* for upper bounds can be obtained from CTE (see Chapter **??**) by replacing step 2 of its main loop and the final part of computing the upper bounds on the joint belief by the procedure given in Figure **??** yielding the Algorithm in Figure 1.16. The partitioning of clusters to mini-clusters can be done in an identical manner to partitioning buckets into mini-buckets.

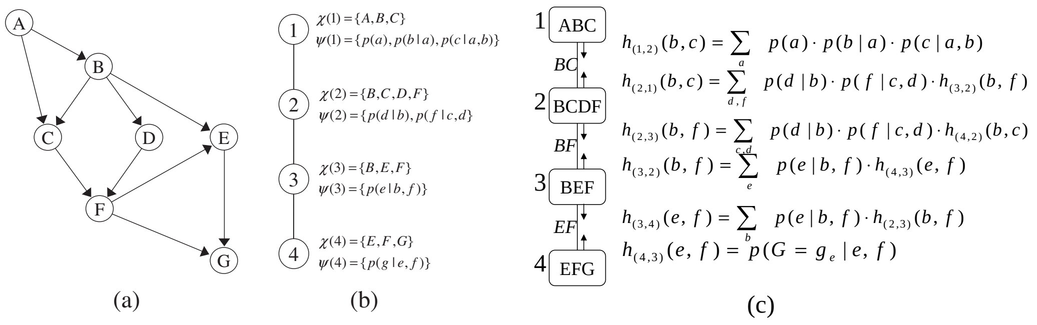
Example 8.15 Figure 1.18 shows the trace of running WMC-bel(3) with all weights w=0 except for one mini-cluster, on the problem in Figure 1.17. First, evidence $G = g_e$ is assigned in all CPTs. There are no individual functions to be sent from cluster 1 to cluster 2. Cluster 1 contains only 3 variables, $\chi(1) = \{A, B, C\}$, therefore it is not partitioned. The combined function $h^1_{1\to2}(b,c) = \sum_a p(a) \cdot p(b|a) \cdot p(c|a,b)$ is computed and the message $H_{1\to2} = \{h^1_{1\to2}(b,c)\}$ is sent to node 2. Now, node 2 can send its message to node 3. Cluster 2 contains 4 variables, $\chi(2) = \{B, C, D, F\}$, and a partitioning is necessary: MC-bel(3) can choose $mc(1) = \{p(d|b), h_{1\to2}(b,c)\}$ and $mc(2) = \{p(f|c,d)\}$. The combined functions using power sum is $h^1_{2\to3}(b) = (\sum_{c,d}(p(d|b) \cdot h_{1\to2}(b,c))^{\frac{1}{w_1}})^{w_1}$ and $h^2_{2\to3}(f) = (\sum_{c,d}(p(f|c,d))^{\frac{1}{w_2}})^{w_2}$ where $w_1 + w_2 = 1$ are computed and the message $H_{4\to3} = \{h^1_{2\to3}(b), h^2_{2\to3}(f)\}$ 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

Weighted Mini-Clustering Elimination for Belief Updating (WMC-bel(i))
Input: A tree decomposition $\langle T, \chi, \psi \rangle T = (V, E)$ for
$\mathcal{B} = \langle X, D, G, P \rangle$. Evidence 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 evidence:**
  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 $cluster_v(u)$, $\{mc(1), \ldots, mc(p)\}$; select weights $w_1, ..., w_p$, $\sum_{i=1}^{p} w_i = 1$.
- $h^j_{u \to v} = (\sum_{elim(u,v)} \prod_{f \in mc(j)} f^{\frac{1}{w_j}})^{w_j}$
- add $\{h^j_{u \to v} | j = 1, \ldots, p\}$ to $H_{u \to v}$. Send the set of messages $H_{u \to v}$ to v.

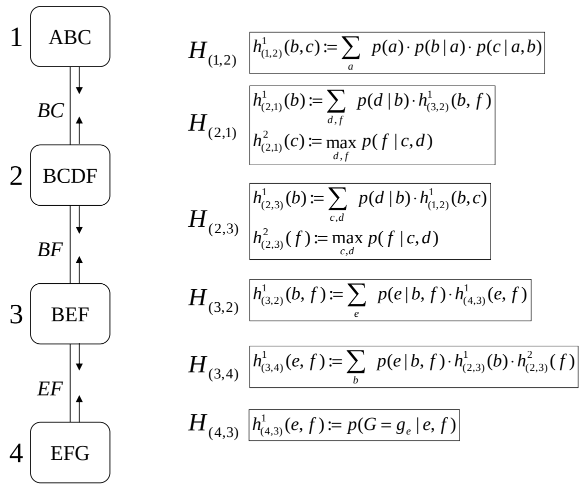
2. **Compute upper bounds** $U(X_i, e)$ **on** $P(X_i, e)$**:**
For every $X_i \in X$ let $u \in V$ be a cluster such that $X_i \in \chi(u)$. Apply (i)-partition mini-clusters for $cluster(u)$, $\{mc(1), \ldots, mc(p)\}$; Compute $\overline{P}(X_i, e) = \prod_{k=1}^{p} (\sum_{\chi(u) - X_i} (\prod_{f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

Figure 8.16: Procedure Mini-Clustering for Belief Updating (WMC-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) = Σ_a p(a)·p(b|a)·p(c|a,b)
h_(2,1)(b,c) = Σ_{d,f} p(d|b)·p(f|c,d)·h_(3,2)(b,f)
h_(2,3)(b,f) = Σ_{c,d} p(d|b)·p(f|c,d)·h_(4,2)(b,c)
h_(3,2)(b,f) = Σ_e p(e|b,f)·h_(4,3)(e,f)
h_(3,4)(e,f) = Σ_b p(e|b,f)·h_(2,3)(b,f)
h_(4,3)(e,f) = p(G = g_e|e,f)
(c)

Figure 8.17: (a) A belief network; (b) A join-tree decomposition; (c) Execution of CTE-bel.



1 ABC
BC
2 BCDF
BF
3 BEF
EF
4 EFG
H_(1,2) h^1_(1,2)(b,c) := Σ_a p(a)·p(b|a)·p(c|a,b)
H_(2,1) h^1_(2,1)(b) := Σ_{d,f} p(d|b)·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) := Σ_{c,d} p(d|b)·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) := Σ_e p(e|b,f)·h^1_(4,3)(e,f)
H_(3,4) h^1_(3,4)(e,f) := Σ_b p(e|b,f)·h^1_(2,3)(b)·h^2_(2,3)(f)
H_(4,3) h^1_(4,3)(e,f) := p(G = g_e|e,f)

Figure 8.18: Execution of MC-bel for $i = 3$

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, WMC-bel(3) never processes more than 3 variables at a time. It is easy to see that,

Theorem 8.16 *Given a Bayesian network $\mathcal{B} = \langle X, D, G, P\rangle$, WMC-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 [**?**])

Theorem 8.17 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.*

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 [**?**]. 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 [**?**] 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).

CHAPTER 9

Bounding Inference by Iterative message-passing schemes

As noted, the BTE algorithm, when applied to singly-connected problems becomes a very efficient propagation algorithm that solves the queries (marginal or mpe) exactly. This is why it is called belief propagation. The algorithm is well defined even if the graph is not tree-structured. When the graphical model is not tree-structured, then BP is no longer exact. If we initialize all of the messages in our system to some fixed value, e.g. the value "1", $\lambda_{i \to j} = 1$, then the message updates are still valid. Iteratively applying these message updates on the dual graph yields the approximate inference method known as Loopy Belief Propagation (BP, or Iterative Belied propagation (IBP)). IBP is an iterative method that is not guaranteed to converge. And, when the algorithm does converge, the resulting beliefs may poorly approximate the true marginals, in the case of the marginal inference task, or give a poor approximation to the mpe assignment, in the case of the mpe inference task. However, despite this lack of guarantees the algorithms performs quite well in many cases (e.g., coding networks) and in other cases.

This section discusses Iterative bounded inference algorithms such as iterative belief propagation and iterative Join-Graph Propagation. One motivation for designing this algorithm is to combine the anytime feature of Mini-Clustering (MC) and the iterative virtues of Iterative Belief Propagation (IBP).

9.1 ITERATIVE JOIN-GRAPH PROPAGATION

Mini-clustering is not fully an anytime algorithm because it is not anyspace. Namely it is limited by the available memory. It works on tree-decompositions and it converges in two passes, so iterating doesn't change the messages. IBP is an iterative algorithm that converges in many cases, and when it converges it does so very fast. Allowing it more time doesn't improve its accuracy. The immediate question is if we can exploit the anytime property of MC obtained using its i-bound, with the iterative qualities of IBP. Algorithm Iterative Join-graph Propagation (IJGP) was designed to benefit from both these directions. It works on a general join-graph which may contain cycles. The cluster size of the graph is user adjustable by the *i-bound* parameter.

The algorithm applies message computation over a join-graph decomposition, which has all the ingredients of a join-tree, except that the underlying graph may have cycles.

Definition 9.1 join-graph decompositions. A *join-graph decomposition* for $BN =< X, D, G, P >$ is a triple $D =< JG, \chi, \psi >$, where $JG = (V, E)$ is a graph, and χ and ψ are labeling functions which associate with each vertex $v \in V$ two sets, $\chi(v) \subseteq X$ and $\psi(v) \subseteq P$ such that:

1. For each $p_i \in P$, there is *exactly* one vertex $v \in V$ such that $p_i \in \psi(v)$, and $scope(p_i) \subseteq \chi(v)$.
2. (connectedness) For each variable $X_i \in X$, the set $\{v \in V | X_i \in \chi(v)\}$ induces a connected subgraph of G, a property also called the running intersection property.

We will refer to a node and its CPT functions as a *cluster* (note that a node may be associated with an empty set of CPTs) and use the term *join-graph-decomposition* and *cluster graph* interchangeably. A *join-tree-decomposition* or a *cluster tree* is the special case when the join-graph *JG* is a tree.

It is clear that one of the problems of message propagation over cyclic join-graphs is *overcounting*. Various schemes were proposed to overcome this problem. We will describe here a scheme that avoids cycles relative to a single variable. Recall that in a dual graph some arcs may be redundant and can be removed, sometime yielding a join-tree. We next define the notion arc-minimality that captures this concept.

Definition 9.2 arc-minimality. A join-graph decomposition D is *arc-minimal* if none of its arcs can be removed while still satisfying the connectedness property of Definition 2.1.

If a join-graph decomposition is not arc-minimal it is easy to remove some of its arcs until it becomes arc-minimal. However, the property of arc-minimality is *not* sufficient to ensure variable-based acyclicity though. What is required is that, for every variable X, the arc-subgraph that contains X be a tree.

Example 9.3 The example in Figure 2.1a shows an arc minimal join-graph which contains a cycle relative to variable 4, with arcs labeled with separators. Notice however that if we remove variable 4 from the label of one arc we will have no cycles (relative to single variables) while the connectedness property will still be maintained.

We next refine the definition of join-graph decompositions, when arcs can be labeled with a subset of their separator.

Definition 9.4 ((minimal) arc-labeled join-graph decompositions.) An *arc-labeled decomposition* for $BN =< X, D, P_G, \prod >$ is a graph, $G = (V, E)$, where χ and ψ associate with each vertex $v \in V$ the sets $\chi(v) \subseteq X$ and $\psi(v) \subseteq P$ and θ associates each edge $(v, u) \subset E$ with a set $\theta((v, u)) \subseteq X$ such that:

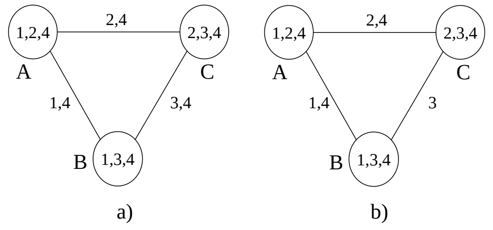


Figure 9.1: An arc-labeled decomposition

1. For each function $p_i \in P$, there is *exactly* one vertex $v \in V$ such that $p_i \in \psi(v)$, and $scope(p_i) \subseteq \chi(v)$.
2. (arc-connectedness) For each arc (u, v), $\theta(u, v) \subseteq sep(u, v)$, such that $\forall X_i \in X$, any two clusters containing X_i can be connected by a path whose every arc's label includes X_i.

Finally, an arc-labeled join-graph is *minimal* if no variable can be deleted from any label while still satisfying the arc-connectedness property.

Recall the following definitions of separators and eliminators.

Definition 9.5 separator, eliminator. Given two adjacent vertices u and v of JG, the *separator* of u and v is defined as $sep(u, v) = \chi(u) \cap \chi(v)))$, and the *eliminator* of u with respect to v is $elim(u, v) = \chi(u) - ((u, v))$. We denote also $\theta(u, v) = sep(u, v)$, where θ are the label on the arcs.

Arc-labeled join-graphs can be made *label-minimal* by removing variables from their labels while maintaining connectedness (if an arc label becomes empty, the arc can be deleted altogether). It is easy to see that,

Proposition 9.6 *A* minimal arc-labeled join-graph *does not contain any cycle relative to any single variable. That is, any two clusters containing the same variable are connected by exactly one path labeled with that variable.*

Notice that every minimal arc-labeled join-graph is arc-minimal (no arc can be deleted), but not vice-versa. The mini-clustering approximation presented in Chapter 3, works by relaxing the join-tree requirement of exact inference into a collection of join-trees having smaller cluster sizes. As we explained, it introduces some independencies in the original problem via node duplication and applies exact inference on the relaxed model requiring only 2 phases of message passings along the mini-cluster tree. For the iterative class of algorithms which we call *IJGP*, we relax the tree-structure requirement and allow *join-graphs*. facilitating iterative message-passing on the resulting cyclic structure. The intuition is that having larger clusters can improve the accuracy of iterative propagation schemes. The join-graphs must not introduce new independencies as we explain next.

We allow only join-graph of a belief network that does not introduce any new independencies to the problem, namely those that are I-map (independency map [109]) relative to node-separation, of the underlying probability distribution. Since we plan to use minimally arc-labeled join-graphs to address over-counting problems, the question is what kind of independencies are captured by such graphs.

Definition 9.7 (edge-separation in (arc-labeled) join-graphs) Let $D = \langle JG, \chi, \psi, \theta \rangle$, $JG = (V, E)$ be an edge-labeled decomposition of a Bayesian network $\mathcal{B} = \langle X, D, P_G, \prod \rangle$. Let $N_W, N_Y \subseteq V$ be two sets of nodes, and $E_Z \subseteq E$ be a set of edges in JG. Let W, Y, Z be their corresponding sets of variables ($W = \cup_{v \in N_W} \chi(v)$, $Z = \cup_{e \in E_Z} \theta(e)$). We say that E_Z *edge-separates* N_W and N_Y in D if there is no path between N_W and N_Y in the JG graph whose edges in E_Z are removed. In this case we also say that W is *separated* from Y given Z in D, and write $\langle W|Z|Y \rangle_D$. Edge-separation in a regular join-graph is defined relative to its separators.

We can show that (see [99].)

Theorem 9.8 *Any arc-labeled join-graph decomposition $D = \langle JG, \chi, \psi, \theta \rangle$ of a belief network $\mathcal{B} = \langle X, D, G, P \rangle$ is an I-map of P relative to edge-separation. Namely, any edge separation in D corresponds to conditional independence in P.*

9.1.1 ALGORITHM IJGP

Applying CTE iteratively to minimal arc-labeled join-graphs yields algorithm *Iterative Join-Graph Propagation (IJGP)* described in Figure 2.2. One iteration of the algorithm applies message-passing in a topological order over the join-graph, forward and back. When node u sends a message (or messages) to a neighbor node v it operates on all the CPTs in its cluster and on all the messages sent from its neighbors excluding the ones received from v. First, all individual functions that share no variables with the eliminator are collected and sent to v. All the rest of the functions are *combined* in a product and summed over the eliminator between u and v.

It is straightforward to show that:

Theorem 9.9

1. *If $IJGP$ is applied to a join-tree decomposition it reduces to join-tree clustering and it therefore is guaranteed to compute the exact beliefs in one iteration.*
2. *The time complexity of one iteration of IJGP is $O(deg \cdot (n + N) \cdot k^{w+1})$ and its space complexity is $O(N \cdot k^{\theta})$, where deg is the maximum degree of a node in the join-graph, n is the number of variables, N is the number of nodes in the graph decomposition, k is the maximum domain size, w is the maximum cluster size and θ is the maximum label size [99].*

```
Algorithm Iterative Join Graph Propagation (IJGP)

Input: An arc-labeled join-graph decomposition $\langle JG, \chi, \psi, \theta \rangle, JG = (V, E)$ for $\mathcal{B} = \langle X, D, P_G, \prod \rangle$. Evidence variables $var(e)$, evidence $e$.
Output: An augmented graph whose nodes are clusters containing the original CPTs and the messages received from neighbors. Approximations of $P(X_i, e),\ \forall X_i \in X$.

Denote by $h_{(u \to v)}$ the message from vertex $u$ to $v$, $ne_v(u)$ the neighbors of $u$ in $JG$ excluding $v$.
$cluster(u) = \psi(u) \cup \{h_{(v \to u)} | (v, u) \in E\}$.
$cluster_v(u) = cluster(u)$ excluding message from $v$ to $u$.

• One iteration of IJGP:
For every node $u$ in $JG$ in some topological order $d$ and back, do
    1. Process evidence variables:
       Assign relevant evidence to all $p_i \in \psi(u)$ $\chi(u) := \chi(u) - var(e),\ \ \forall u \in V$
    2. Compute individual functions:
       Include in $H_{(u \to v)}$ each function in $cluster_v(u)$ whose scope does not contain variables in $elim(u, v)$. Denote by $A$ the remaining functions.
    3. Compute combined function: $h_{(u \to v)} = \alpha \sum_{elim(u,v)} \prod_{f \in A} f$, add to $H_{(u \to v)}$.
       Send $H_{(u \to v)}$ to node $v$.
Endfor
• Compute $P(X_i, e)$:
    For every $X_i \in X$ let $u$ be a vertex in $JG$ such that $X_i \in \chi(u)$.
    Compute $P(X_i, e) = \alpha \sum_{\chi(u) - \{X_i\}} (\prod_{f \in cluster(u)} f)$
```

Figure 9.2: Algorithm Iterative Join-Graph Propagation (IJGP)

Proof. The number of cliques in the chordal graph G' corresponding to G is at most n, so the number of nodes in the join-tree is at most n. The complexity of processing a node u in the join-tree is $deg_u \cdot (|\psi(u)| + deg_u - 1) \cdot k^{|\chi(u)|}$, where deg_u is the degree of u. By bounding deg_u by deg, $|\psi(u)|$ by n and $\chi(u)$ by $w^* + 1$ and knowing that $deg < N$, by summing over all nodes, we can bound the entire time complexity by $O(deg \cdot (n + N) \cdot k^{w^*+1})$.

For each edge IJGP records functions. Since the number of edges in bounded by n and the size of each message is bounded by d^{sep} we get space complexity of $O(n \cdot d^{\theta})$. □

One question which we did not address at all in this section is why propagating the messages iteratively should help. Why is IJGP upon convergence, superior to IJGP with one iteration and is superior to MC? One clue can be provided when considering deterministic constraint networks which can be viewed as "extreme probabilistic networks". It is known that constraint propagation algorithms, which are analogous to the messages sent by belief propagation, are guaranteed to converge and are guaranteed to improve with convergence. The propagation scheme presented here works like constraint propagation relative to the flat network abstraction of P (where all non-zero entries are normalized to a positive constant), and propagation is guaranteed to be more accurate for that abstraction at least. Another explanation is provided [75] by showing a connection between the probability distribution generated by IJGP (upon convergence) and the distribution that minimize a distance function to the exact distribution.

9.1.2 ITERATIVE BELIEF PROPAGATION

Iterative belief propagation (IBP) is an iterative application of Pearl's algorithm for poly-trees [109] to any Bayesian network. We will describe IBP as an instance of join-graph propagation over a *dual graph*. We recap the definition of a dual graph.

Definition 9.10 dual graphs. Given a set of functions $F = \{f_1, ..., f_l\}$ over scopes $S_1, ..., S_l$, *the dual graph* of F is a graph $DG = (V, E, L)$ that associates a node with each function, namely $V = F$ and an edge connects any two nodes whose function's scope share a variable, $E = \{(f_i, f_j)|S_i \cap S_j \neq \phi\}$. L is a set of labels for the arcs, each being labeled by the shared variables of its nodes, $L = \{l_{ij} = S_i \cap S_j|(i, j) \in E\}$. A *dual join-graph* is an arc-labeled edge subgraph of DG. A *minimal* dual join-graph is a dual join-graph for which none of the edge labels can be further reduced while maintaining the connectedness property.

Interestingly, there may be many minimal dual join-graphs of the same dual graph. We will define Iterative Belief Propagation on a dual join-graph. Each node sends a message over an edge whose scope is identical to the label on that edge. Since Pearl's algorithm sends messages whose scopes are singleton variables only, we highlight minimal singleton-label dual join-graphs.

Proposition 9.11 *Any Bayesian network has a minimal dual join-graph where each arc is labeled by a single variable.*

Proof. Consider a topological ordering of the nodes in the acyclic directed graph of the Bayesian network $d = X_1, ..., X_n$. We define the following dual join-graph. Every node in the dual graph $\mathcal{D}$, associated with CPT p_i is connected to node of CPT p_j, $j < i$ if $X_j \in pa(X_i)$. We label the arc between p_j and p_i by variable X_j, namely $l_{ij} = \{X_j\}$. It is easy to see that the resulting arc-labeled subgraph of the dual graph satisfies connectedness. The resulting labeled graph is a dual graph with singleton labels. □

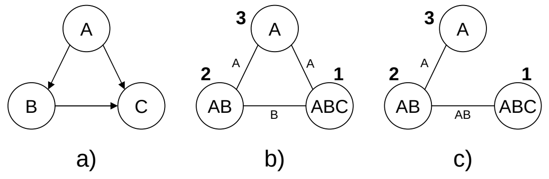


Figure 9.3: a) A belief network; b) A dual join-graph with singleton labels; c) A dual join-graph which is a join-tree

Example 9.12 Consider the belief network on 3 variables A, B, C with CPTs 1. $P(C|A,B)$, 2. $P(B|A)$ and 3. $P(A)$, given in Figure 2.3a. Figure 2.3b shows a dual graph with singleton labels on the edges. Figure 2.3c shows a dual graph which is a join-tree, on which belief propagation can solve the problem exactly in one iteration (two passes up and down the tree).

For completeness, we present algorithm IBP in Figure 2.4 which is a special case of IJGP. When IBP is time and space linear in the size of the belief network. It is also easy to show that when IBP is applied to a minimal singleton-labeled dual graph it coincides with Pearl's belief propagation applied directly to the acyclic graph representation. Also, when the dual join-graph is a tree, IBP converges after one iteration (two passes, up and down the tree) yielding the exact beliefs.

9.1.3 BOUNDED JOIN-GRAPH DECOMPOSITIONS

Since we want to control the complexity of join-graph algorithms, we will define it on decompositions having bounded cluster size. If the number of variables in a cluster is bounded by i, the time and space complexity of processing one cluster is exponential in i. We will call the maximal cluster size in a join-graph, *joinwidth*.

Given a join-graph decomposition $D = \langle JG, \chi, \psi, \theta \rangle$, the joinwidth of D defined as $max_{v \in V} |\chi(v)|$. Intuitively, the accuracy of a typical algorithm in this class the accuracy depends on how far the join-graph is from a join-tree, which may be captured by the treewidth of JG which we would call *external width*. We can now state our target decomposition as follows. Given a graph G, and a bounding parameter i we wish to find a join-graph decomposition D of G whose joinwidth is bounded by i and whose external width is minimized.

We can consider two classes of algorithms. One class is *partition-based*. Algorithms in this class start from a given tree-decomposition and then partitions the clusters until the decomposition has clusters bounded by i. An alternative approach is *grouping-based* algorithms. It starts from a minimal dual-graph-based join-graph decomposition (where each cluster contains a single CPT) and groups clusters into larger clusters as long as the resulting clusters do not exceed the given bound. In both methods one should attempt to reduce the external width of the generated graph-decomposition.

We will next present a partition-based approach which is based on the decomposition suggested by the mini-bucket scheme. Given a bound i, algorithm *Join-Graph Structuring(i)* applies the

```
Algorithm Iterative Belief Propagation (IBP)
Input: An edge-labeled dual join-graph $DG = (V, E, L)$ for a Bayesian network $\mathcal{B} = \langle X, D, P_G, \prod \rangle$. Evidence $e$.
Output: An augmented graph whose nodes include the original CPTs and the messages received from neighbors. Approximations of $P(X_i, e)$, $\forall X_i \in X$. Approximations of $P(F_i, e)$, $\forall F_i \in \mathcal{B}$.
Denote by: $h_{u \to v}$ the message from $u$ to $v$; $ne(u)$ the neighbors of $u$ in $V$; $ne_v(u) = ne(u) - \{v\}$; $l_{uv}$ the label of $(u, v) \in E$; $elim(u, v) = scope(u) - scope(v)$.
• One iteration of IBP
  For every node $u$ in $DJ$ in a topological order and back, do:
    1. Process observed variables
       Assign evidence variables to the each $p_i$ and remove them from the labeled edges.
    2. Compute and send to $v$ the function:
       $$h_{u \to v} = \alpha \sum_{elim(u,v)} (p_u \cdot \prod_{\{h_{i \to u}, i \in ne_v(u)\}} h_{i \to v})$$
  Endfor
• Compute approximations of $P(F_i, e)$, $P(X_i, e)$:
  For every $X_i \in X$ let $u$ be the vertex of family $F_i$ in $DJ$,
  $P(F_i, e) = \alpha (\prod_{h_{i \to u}, u \in ne(i)} h_{i \to u}) \cdot p_u$;
  $P(X_i, e) = \alpha \sum_{scope(u) - \{X_i\}} P(F_i, e)$.
```

Figure 9.4: Algorithm Iterative Belief Propagation

procedure *Schematic Mini-Bucket(i)*, described in Figure 2.6. The procedure only traces the scopes of the functions that would be generated by the full mini-bucket procedure, avoiding actual function computation. The procedure ends with a collection of mini-bucket trees, each rooted in the mini-bucket of the first variable. Each of these trees is minimally edge-labeled. Then, *in-edges* labeled with only one variable are introduced, and they are added only to obtain the running intersection property between branches of these trees.

Proposition 9.13 *Algorithm Join-Graph Structuring(i) generates a minimal edge-labeled join-graph decomposition having bound i.*

Proof. The construction of the join-graph specifies the vertices and edges of the join-graph, as well as the variable and function labels of each vertex. We need to demonstrate that 1) the connectedness property holds, and 2) that edge-labels are minimal.

```
Algorithm Join-Graph Structuring(i)
  1. Apply procedure schematic mini-bucket(i).
  2. Associate each resulting mini-bucket with a node in the join-graph, the variables of the nodes are those appearing in the mini-bucket, the original functions are those in the mini-bucket.
  3. Keep the edges created by the procedure (called out-edges) and label them by the regular separator.
  4. Connect the mini-bucket clusters belonging to the same bucket in a chain by in-edges labeled by the single variable of the bucket.
```

Figure 9.5: Algorithm Join-Graph Structuring(i).

```
Procedure Schematic Mini-Bucket(i)
  1. Order the variables from $X_1$ to $X_n$ minimizing (heuristically) induced-width, and associate a bucket for each variable.
  2. Place each CPT in the bucket of the highest index variable in its scope.
  3. For $j = n$ to 1 do:
     Partition the functions in $bucket(X_j)$ into mini-buckets having at most $i$ variables.
     For each mini-bucket $mb$ create a new scope-function (message) $f$ where $scope(f) = \{X | X \in mb\} - \{X_i\}$ and place scope(f) in the bucket of its highest variable. Maintain an edge between $mb$ and the mini-bucket (created later) of $f$.
```

Figure 9.6: Procedure Schematic Mini-Bucket(i).

Connectedness property specifies that for any 2 vertices u and v, if vertices u and v contain variable X, then there must be a path $u, w_1, ..., w_m, v$ between u and v such that every vertex on this path contains variable X. There are two cases here. 1) u and v correspond to 2 mini-buckets in the same bucket, or 2) u and v correspond to mini-buckets in different buckets. In case 1 we have 2 further cases, 1a) variable X is being eliminated in this bucket, or 1b) variable X is not eliminated in this bucket. In case 1a, each mini-bucket must contain X and all mini-buckets of the bucket are connected as a chain, so the connectedness property holds. In case 1b, vertexes u and v connect to their (respectively) parents, who in turn connect to their parents, etc. until a bucket in the scheme is reached where variable X is eliminated. All nodes along this chain include variable X, so the connectedness property holds. Case 2 resolves like case 1b.

To show that edge labels are minimal, we need to prove that there are no cycles with respect to edge labels. If there is a cycle with respect to variable X, then it must involve at least one in-edge (edge connecting two mini-buckets in the same bucket). This means variable X must be the variable being eliminated in the bucket of this in-edge. Therefore, variable X is not contained in any of the parents of the mini-buckets of this bucket. Therefore, in order for the cycle to exist, another in-edge down the bucket-tree from this bucket must contain X. However, this is impossible as this would imply that variable X is eliminated twice. □

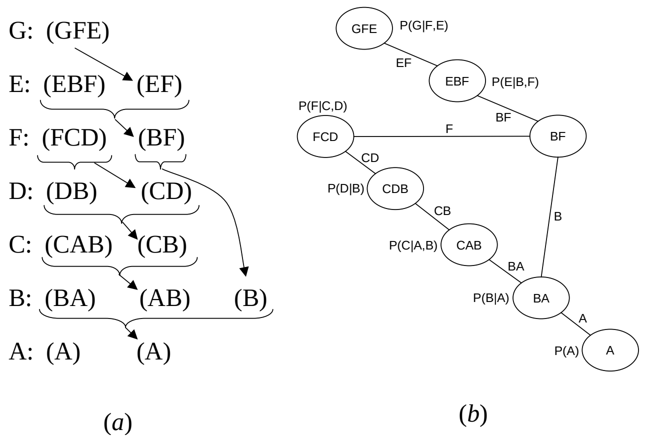


Figure 9.7: Join-graph decompositions.

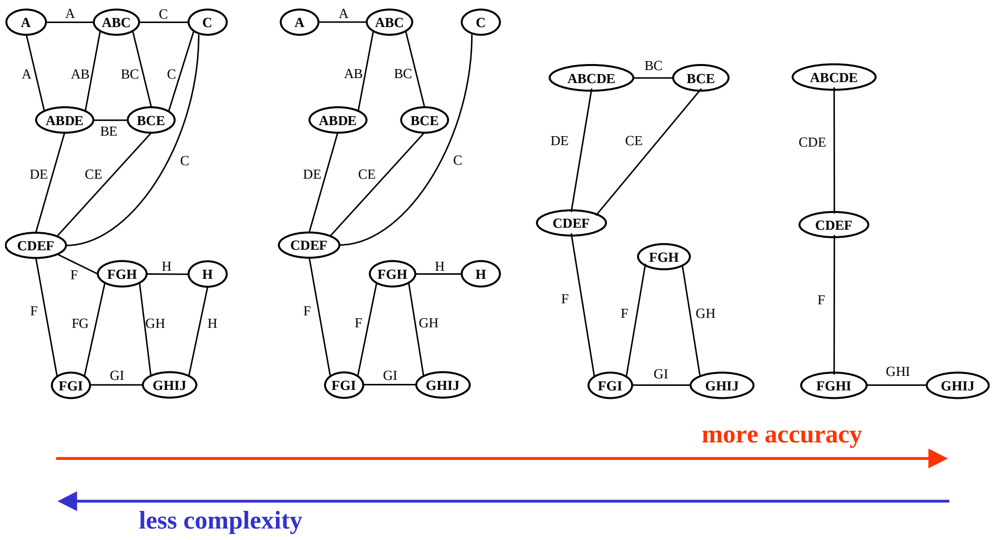


Figure 9.8: Join-graphs.

Example 9.14 Figure 2.7a shows the trace of procedure schematic mini-bucket(3) applied to the problem described in Figure 1.17a. The decomposition in Figure 2.7b is created by the algorithm graph structuring. The only cluster partitioned is that of F into two scopes (FCD) and (BF), connected by an in-edge labeled with F.

Example 9.15 Figure 2.8 shows a range of edge-labeled join-graphs. On the left extreme we have a graph with smaller clusters, but more cycles. This is the type of graph IBP works on. On the right extreme we have a tree decomposition, which has no cycles but has bigger clusters. In between, there could be a number of join-graphs where maximum cluster size can be traded for number of cycles. Intuitively, the graphs on the left present less complexity for join-graph algorithms because the cluster size is small, but they are also likely to be less accurate. The graphs on the right side are computationally more complex, because of larger cluster size, but are likely to be more accurate.

9.2 COST-SHIFTING FOR BETTER APPROXIMATION

All the decomposition bounds that we have shown can be improved, or tightened by what is sometime called *cost-shifting* or *re-parameterization* and fall under the class of variational schemes. The idea is to shift some costs from one local function to another in a way that does not change the global function, but can improve the decomposition bound. We will see several variants of these ideas, one is mini-bucket elimination with max-marginal matching (MBE-MM), which is a non-iterative algorithm that applies a single pass of cost-shifting during the mini-bucket construction bucket by bucket. The second approach, Join Graph Linear Programming (JGLP), iteratively applies cost-shifting updates to the full mini-bucket join-graph.

9.2.1 LINEAR PROGRAMMING METHODS

To introduce the idea assume for simplicity that the network consists only of pairwise functions $F = \{f_{ij}(X_i, X_j)\}$ and that the problem is max-sum, i.e., the task is to compute $C^* = \max_{\mathbf{X}} \sum_{f_{ij} \in F} f_{ij}(X_i, X_j)$. A simple bound on the max-sum objective is then given by maxima of the individual functions, exchanging the sum and max operators, as is common in the mini-bucket scheme:

$$C^* = \max_{\mathbf{X}} \sum_{f_{ij} \in F} f_{ij}(X_i, X_j) \leq \sum_{f_{ij} \in F} \max_{X_i, X_j} f_{ij}(X_i, X_j) \tag{9.1}$$

As noted for the mini-bucket case we can interpret this operation as making an individual copy of each variable for each function, and optimizing over them separately. See, for example, the problem in Figure 2.9(a) with 3 variables and 3 functions. Figure 2.9(e) shows the modified problem with each variable duplicated, so that each function can be maximized over independently. We can also derive a bound on the optimal cost by introducing a collection of functions $\{\lambda_{ij}(X_i), \lambda_{ji}(X_j)\}$ for each edge (ij) and requiring

$$\lambda \in \Lambda \quad \Leftrightarrow \quad \forall i, \quad \sum_j \lambda_{ij}(X_i) = 0 \tag{9.2}$$

Then, we have

$$\begin{aligned} C^* &= \max_{\mathbf{X}} \sum_{f_{ij} \in F} f_{ij}(X_i, X_j) \\ &= \max_{\mathbf{X}} \sum_{f_{ij} \in F} f_{ij}(X_i, X_j) + \sum_i \sum_j \lambda_{ij}(X_i) \\ &\leq \min_{\lambda \in \Lambda} \sum_{f_{ij} \in F} \max_{X_i, X_j} \big(f_{ij}(X_i, X_j) + \lambda_{ij}(X_i) + \lambda_{ji}(X_j) \big) \end{aligned} \tag{9.3}$$

The last expression is obtained by distributing each λ_{ij} to its associated factor and applying the inequality (2.1).

The new functions $\tilde{f}_{ij} = f_{ij}(X_i, X_j) + \lambda_{ij}(X_i) + \lambda_{ji}(X_j)$ define a *re-parametrization* of the original distribution, i.e., they change the individual functions without modifying the global function $F(\mathbf{X}) = \sum_{\mathbf{X}} f_{ij} = \sum_{\mathbf{X}} \tilde{f}_{ij}$. The λ_{ij} can be interpreted as "cost-shifting" operations that transfer cost from one function to another while preserving the overall cost [3] or as Lagrange multipliers enforcing consistency among the copies of X_i [2, 75]. In the former interpretation, the updates are called "soft arc-consistency" due to their similarity to arc-consistency for constraint satisfaction. Under the latter view, the bound corresponds to a *dual decomposition* solver for a linear programming (LP) relaxation of the original problem.

The main distinguishing feature among such dual decomposition approaches is the way in which the bound is tightened by updating the functions λ. This is done either by sub-gradient approaches or by coordinate descent updates that can be interpreted as "message passing" [**? ?**]. Without going into the details of these approaches, we refer to these iterative bound improvement updates as "LP-tightening" updates.

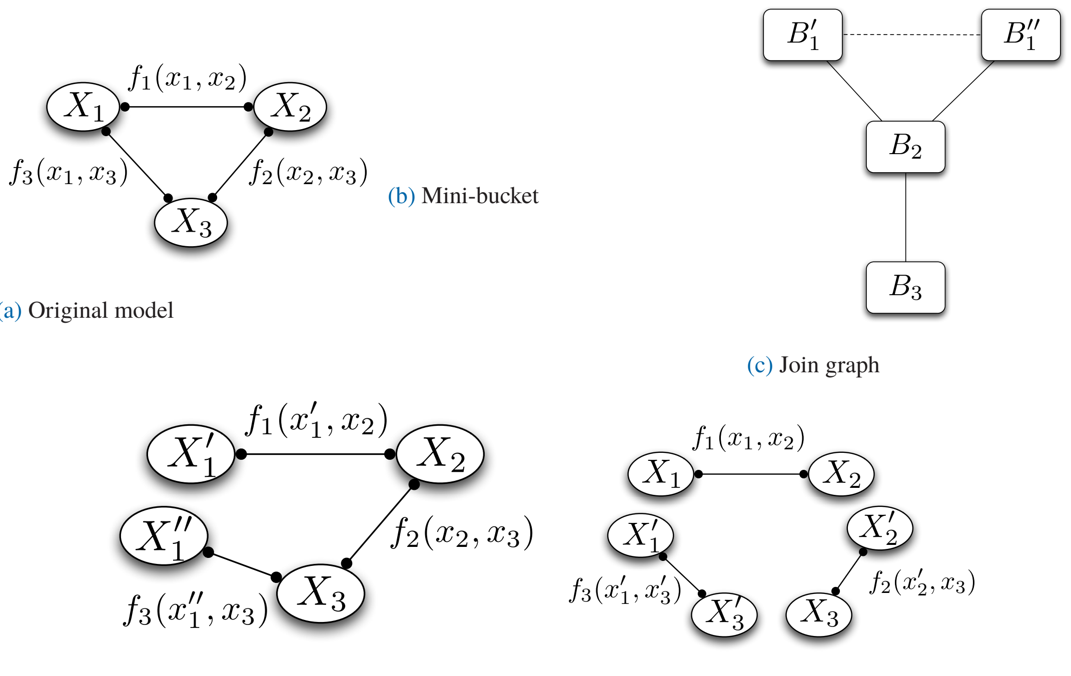
LP-tightening algorithm. Let's see a derivation of a particular simple, yet effective scheme that minimizes over λs the expression

$$\sum_{f_{ij} \in F} \max_{X_i, X_j} \big(f_{ij}(X_i, X_j) + \lambda_{ij}(X_i) + \lambda_{ji}(X_j) \big) \tag{9.4}$$

thus tightening the upper bound on C^* (Eq. 2.3).

The LP-tightening scheme is initialized with all $\lambda_{ij}(X_i) = 0$. In the spirit of well-known coordinate descent approaches we iteratively minimize expression 2.4 with respect to each $\lambda_{ij}(X_i)$ separately, while fixing the values of all other λs. For simplicity of derivation, without loss of generality, lets assume that each variable X_i appears in the scope of at most two functions, $f_{ij}(X_i, X_j)$ and $f_{ik}(X_i, X_k)$. Identifying only the terms relevant to variable X_i, we get:

$$\begin{aligned} &\min_{\lambda_{ij}(X_i), \lambda_{ik}(X_i)} \Big[\big[\max_{X_i, X_j} f_{ij}(X_i, X_j) + \lambda_{ij}(X_i) \big] + \big[\max_{X_i, X_k} f_{ik}(X_i, X_k) + \lambda_{ik}(X_i) \big] \Big] \\ &= \min_{\lambda_{ij}(X_i), \lambda_{ik}(X_i)} \Big[\max_{X_i} \big[\max_{X_j} f_{ij}(X_i, X_j) + \lambda_{ij}(X_i) \big] + \max_{X_i} \big[\max_{X_k} f_{ik}(X_i, X_k) + \lambda_{ik}(X_i) \big] \Big] \end{aligned}$$



(a) Original model

(b) Mini-bucket

(c) Join graph

(d) Model with duplicated variables in a mini-bucket

(e) Model with all variables duplicated

Figure 9.9: The mini-bucket procedure for a simple graph. (a) Primal graph; (b) the buckets and messages computed in MBE; (c) join-graph, dotted line corresponds to an edge absent from a mini-bucket tree; (d) interpreting MBE as variable duplication, X_1 is duplicated in each of two mini-buckets $q_1^1 = \{f_1(X_1, X_2)\}$ and $q_1^2 = \{f_3(X_1, X_3)\}$; (e) The graph on which FGLP runs with all variables duplicated and each factor processed separately.

Let us define the so-called "max-marginals" $\gamma_{ij}(X_i) = \max_{X_j} f_{ij}(X_i, X_j)$ and re-arrange the terms in the above expression, yielding the following bound:

$$
\begin{aligned}
&\min_{\lambda_{ij}(X_i),\lambda_{ik}(X_i)} \left[\max_{X_i} \left[\max_{X_j} f_{ij}(X_i, X_j) + \lambda_{ij}(X_i) \right] + \max_{X_i} \left[\max_{X_k} f_{ik}(X_i, X_k) + \lambda_{ik}(X_i) \right] \right] \\
&= \min_{\lambda_{ij}(X_i),\lambda_{ik}(X_i)} \left[\max_{X_i} \left[\gamma_{ij}(X_i) + \lambda_{ij}(X_i) \right] + \max_{X_i} \left[\gamma_{ik}(X_i) + \lambda_{ik}(X_i) \right] \right] \\
&\geq \min_{\lambda_{ij}(X_i),\lambda_{ik}(X_i)} \left[\max_{X_i} \left[\gamma_{ij}(X_i) + \gamma_{ik}(X_i) + \lambda_{ij}(X_i) + \lambda_{ik}(X_i) \right] \right]
\end{aligned}
\tag{9.5}
$$

We force that $\lambda_{ij}(X_i) + \lambda_{ik}(X_i) = 0$ to ensure that the total cost does not changed (Equation 2.2).

Algorithm 6: LP-tightening (based on [?])

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, where $f_{\mathbf{S}_i}$ is a potential defined on variables $\mathbf{S}_i$
Output: Upper bound on the optimum value of $\max_{\mathbf{X}} \sum \mathbf{F}$
while NOT converged do
    for any pair of function scopes $\mathbf{S}_i$, $\mathbf{S}_j$ such that $\mathbf{S}_{ij} = \mathbf{S}_i \cap \mathbf{S}_j \neq \emptyset$ do
        Compute max-marginals:
        $\gamma_{\mathbf{S}_i}(\mathbf{S}_{ij}) = \max_{\mathbf{S}_i \setminus \mathbf{S}_{ij}} f_{\mathbf{S}_i}$;
        $\gamma_{\mathbf{S}_j}(\mathbf{S}_{ij}) = \max_{\mathbf{S}_j \setminus \mathbf{S}_{ij}} f_{\mathbf{S}_j}$;
        Update parametrization:
        $f_{\mathbf{S}_i} \leftarrow f_{\mathbf{S}_i} + \frac{1}{2}\big(\gamma_{\mathbf{S}_j}(\mathbf{S}_{ij}) - \gamma_{\mathbf{S}_i}(\mathbf{S}_{ij})\big)$;
        $f_{\mathbf{S}_j} \leftarrow f_{\mathbf{S}_j} + \frac{1}{2}\big(\gamma_{\mathbf{S}_i}(\mathbf{S}_{ij}) - \gamma_{\mathbf{S}_j}(\mathbf{S}_{ij})\big)$;
```

Many choices of λ_{ij} are known to achieve the minimum of the right-hand side of expression 2.5. Lets select the following one:

$$\lambda_{ij}(X_i) = \frac{1}{2}\big(\gamma_{ik}(X_i) - \gamma_{ij}(X_i)\big) = \frac{1}{2}\big(\max_{X_k} f_{ik}(X_i, X_k) - \max_{X_j} f_{ij}(X_i, X_j)\big) \tag{9.6}$$

Iterating over all functions, while updating each function $\forall i, j, f_{ij}(X_i, X_j) \leftarrow f_{ij}(X_i, X_j) + \lambda_{ij}(X_i)$ and recalculating λ_{ij} at each step until convergence, yields a minimization procedure that can be interpreted as a *max-marginal* or *moment-matching* procedure on the functions $f_{ij}(X_i, X_j)$. Intuitively, we would like the updates to diminish in magnitude and converge to zero as fast as possible. To achieve that, taking into account Equation 2.6, we would like to make the max-marginals $\gamma_{ij}(X_i)$ and $\gamma_{ik}(X_i)$ of each variable X_i equal to each other. Algorithm 1 generalizes this update to higher-order functions $f_{\mathbf{S}_i}$ over scopes of variables $\mathbf{S}_i \subseteq \mathbf{X}$.

A well-known algorithm quite similar to the above LP-tightening in Algorithm 1 is a message-passing scheme called Max-Product Linear Programming (MPLP) [?]. Algorithm 2 presents a version of MPLP that we call Factor Graph Linear Programming (FGLP). At each step FGLP simultaneously updates all functions $f_{ij}(X_i, X_j)$ involving a single variable X_i. The messages sent by the algorithms on a factor graph are schematically illustrated in Figure 2.10. In this example variable X_i is in the scope of three functions: $f_{\mathbf{S}_t}(X_i, X_k, X_m)$, $f_{\mathbf{S}_p}(X_i, X_j)$ and $f_{\mathbf{S}_q}(X_i, X_n)$, where $\mathbf{S}_t = \{X_i, X_k, X_m\}$, $\mathbf{S}_p = \{X_i, X_j\}$ and $\mathbf{S}_q = \{X_i, X_n\}$. Note that in the figure we only show the messages involving X_i. The max-marginals are: $\gamma_{\mathbf{S}_q}(X_i) = \max_{X_n} f_{\mathbf{S}_q}(X_i, X_n)$, $\gamma_{\mathbf{S}_p}(X_i) = \max_{X_j} f_{\mathbf{S}_p}(X_i, X_j)$ and $\gamma_{\mathbf{S}_t}(X_i) = \max_{X_k, X_m} f_{\mathbf{S}_t}(X_i, X_k, X_m)$. The update messages from vari-

Algorithm 7: Factor Graph Linear Programming (FGLP, based on [**?**])

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, , \mathbf{F}, \sum \rangle$, variable ordering $o$
Output: Upper bound on the optimum value of MPE cost
while NOT converged do
    for each variable $X_i$ do
        Get factors $F_i = f_{\mathbf{S}_k} : X_i \in \mathbf{S}_k$ with $X_i$ in their scope;
        //for each function compute max-marginals $\gamma$ marginalizing out all variables except for $X_i$:
        $\forall f_{\mathbf{S}_k}\ \gamma_{\mathbf{S}_k}(X_i) = \max_{\mathbf{S}_k \setminus X_i} f_{\mathbf{S}_k}$;
        // compute messages $\beta_{\mathbf{S}_k}(X_i)$ from $X_i$ back to a function $f_{\mathbf{S}_k}$ correcting for the function's own
        max-marginal $\gamma_{\mathbf{S}_k}$:
        $\forall f_{\mathbf{S}_k}\ \beta_{\mathbf{S}_k} = \frac{1}{|F_i|} \sum_{\{\mathbf{S}_j | f_{\mathbf{S}_j} \in F_i\}} \gamma_{\mathbf{S}_j}(X_i) - \gamma_{\mathbf{S}_k}(X_i)$
        // update (re-parametrize) each function:
        $\forall f_{\mathbf{S}_k}, f_{\mathbf{S}_k} \leftarrow f_{\mathbf{S}_k} + \beta_{\mathbf{S}_k}$;
```

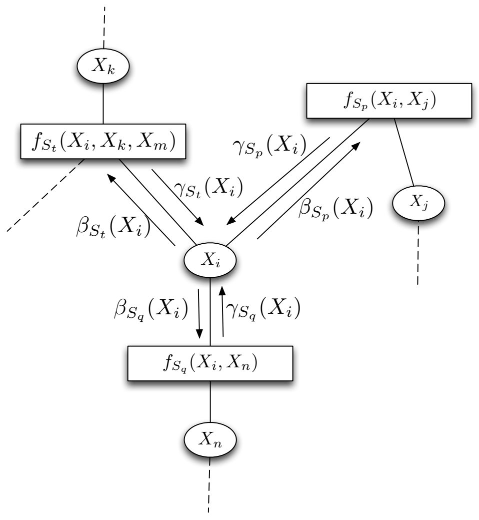


Figure 9.10: FGLP example: local messages involving variable X_i.

able X_i back to the functions are:

$$\beta_{\mathbf{S}_q}(X_i) = \frac{1}{3}(\gamma_{\mathbf{S}_q}(X_i) + \gamma_{\mathbf{S}_p}(X_i) + \gamma_{\mathbf{S}_t}(X_i)) - \gamma_{\mathbf{S}_q}(X_i)$$
$$\beta_{\mathbf{S}_p}(X_i) = \frac{1}{3}(\gamma_{\mathbf{S}_q}(X_i) + \gamma_{\mathbf{S}_p}(X_i) + \gamma_{\mathbf{S}_t}(X_i)) - \gamma_{\mathbf{S}_p}(X_i)$$
$$\beta_{\mathbf{S}_t}(X_i) = \frac{1}{3}(\gamma_{\mathbf{S}_q}(X_i) + \gamma_{\mathbf{S}_p}(X_i) + \gamma_{\mathbf{S}_t}(X_i)) - \gamma_{\mathbf{S}_t}(X_i)$$

Algorithm 8: Algorithm JGLP

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, , \mathbf{F}, \sum \rangle$, variable order $d = \{X_1, \ldots, X_n\}$, parameter $i$.
Output: Upper bound on the optimum value of MPE cost
//Initialize: Partition the functions in F into $\mathbf{B}_1, \ldots, \mathbf{B}_n$, where $\mathbf{B}_p$ contains all functions $f_{S_j}$ whose
highest variable is $X_p$;
Build mini-bucket join graph; // (Algorithm ??)
Find the function of each mini-bucket $q_k^p$:
$F_k^p = \sum_{f_{\mathbf{S}_j} \in q_k^p} f_{\mathbf{S}_j}$
while NOT converged OR NOT time limit reached do
    for all pairs of mini-buckets $q_k^p$, $q_k^l$ connected by an edge do
        Find the separators $S = Scope(q_k^p) \cap Scope(q_k^l)$ ;
        Find the max-marginals of each mini-bucket
        $q_k^p$: $\gamma_k^p = \max_{Scope(q_k^p)-S}(F_k^p)$;
        $q_k^j$: $\gamma_k^l = \max_{Scope(q_k^l)-S}(F_k^l)$;
        Update functions in both mini-buckets
        $q_k^p$: $F_k^p \leftarrow F_k^p - \frac{1}{2}(\gamma_k^p - \gamma_k^l)$
        $q_k^l$: $F_k^l \leftarrow F_k^l + \frac{1}{2}(\gamma_k^p - \gamma_k^l)$;
```

Theorem 9.16 Complexity of FGLP. *The total time complexity of a single iteration of FGLP is $O(n \cdot Q \cdot k^{Sc})$, where n is the number of variables in the problem, k is the largest domain size, $|\boldsymbol{F}|$ is the number of functions, Sc bounds the largest scope of the original functions, Q is the largest number of functions having the same variable X_j in their scopes. The space complexity is $O(|\boldsymbol{F}| \cdot k^{Sc})$.*

9.2.2 JOIN GRAPH LINEAR PROGRAMMING

Join-Graph MBE Structuring. The mini-bucket procedure defines a mini-bucket tree. Each mini-bucket defines a cluster. Two mini-buckets are connected if there exists a message between them. Once the mini-buckets of the same variables are connected, the mini-bucket tree yields a join-graph, where each cluster has at most $i + 1$ variables, where i is the i-bound.

Join-Graph MBE Structuring (Algorithm 2.5) constructs a join-graph using the mini-bucket elimination. The structure is the same as the mini-bucket-tree assuming variable duplication with additional arcs connecting mini-buckets of the same bucket in a line.

Figure 2.9 presents an example of a problem with 3 variables, whose primal graph is shown in Figure 2.9(a). In Figure 2.9(b) we see the trace of MBE on the problem, namely the buckets and the messages computed. Figure 2.9(c) shows the join-graph created by Join-Graph MBE Structuring. The dotted line corresponds to an edge absent from a mini-bucket tree and added during step at line 8.

Join-Graph Linear Programming. From the perspective of linear programming relaxation (Section 2.2.1) mini-bucket elimination can be interpreted as running a single pass of LP-tightening, sending messages top down only along edges of the spanning tree of the mini-bucket join graph. A straightforward extension of MBE can be an iterative procedure that repeatedly performs LP-tightening along all of the join graph edges. Algorithm 3 shows the resulting Join-Graph Linear Programming (JGLP) scheme. It constructs the join graph (line 1), calculated mini-bucket functions F_k^p (line 2) and performs re-parametrization updates to F_k^p (as in Algorithm 1) until convergence (lines 2-7). Note that once JGLP converges, performing mini-bucket elimination on the resulting graph will not change the bound value.

Theorem 9.17 Complexity of JGLP. *Given a problem with n variables with largest domains of size k, where at most Q functions have the same variable in their scope, and i-bound i, the time complexity of a single iteration of JGLP is $O(n \cdot Q \cdot k^i)$. The time complexity of join-graph construction step is $O(n \cdot k^{i+1})$. The overall space complexity is $O(n \cdot k^{i+1} + n \cdot k^i)$.*

Proof. The complexity of constructing a join-tree is the same as the complexity of running full mini-bucket elimination algorithm, namely $O(n \cdot k^{i+1})$. The time complexity of a single iteration of JGLP consists of performing for each edge the following steps: 1. compute max-marginals of a pair of clique functions, requires time equal to $O(2 \cdot k^{i+1-|S|})$, where $|S|$ is the size of a separator between the mini-buckets of the same variable. 2. compute the mean, which takes $O(2 \cdot k^{|S|})$ time. 3. update the clique functions, requiring $O(2 \cdot k^{i+1})$ time.

The space complexity of the algorithm is dominated by the necessity of storing in memory the join-graph, whose size is bounded by $O(n \cdot k^{i+1})$ and the messages between the clusters of size $O(n \cdot k^i)$, yielding the overall space complexity of $O(n \cdot k^{i+1})$. □

9.2.3 MBE-MM

While the iterative nature of JGLP yields more accurate bounds than MBE, in practice it can have a significant additional time and space overhead compared to MBE.

We next present a non-iterative scheme that performs re-parametrization between the mini-buckets of the same variable only. The algorithm mini-bucket elimination with max-marginal matching (MBE-MM, Algorithm 4) proceeds by following the standard mini-bucket downward pass. When each mini-bucket $q_k^p \in Q_k$ is processed, before eliminating variable X_k, we first perform an LP-tightening update to the mini-bucket functions $f_{q_k^p}$. For storage and computational efficiency reasons, we perform a single update on all mini-buckets of the same variable simultaneously, matching their max-marginals on their joint intersection. Alternatively, the updates can be done between all possible pairs of mini-buckets of the same bucket. While not guranteed, it is reasonable to expect

Algorithm 9: Algorithm MBE-MM

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, , \mathbf{F}, \sum \rangle$, variable order $d = \{X_1, \ldots, X_n\}$, i-bound parameter $i$
Output: Upper bound on the optimum value of MPE cost
//Initialize:
Partition the functions in F into $\mathbf{B}_1, \ldots, \mathbf{B}_n$, where $\mathbf{B}_k$ contains all functions $f_j$ whose highest variable is $X_k$;
//processing bucket $\mathbf{B}_k$
for $k \leftarrow n$ down to 1 do
    Partition functions $g$ (both original and messages generated in previous buckets) in $\mathbf{B}_k$ into the mini-buckets
    defined $Q_{X_k} = \{q_k^1, \ldots, q_k^t\}$, where each $q_k^p$ has no more than $i+1$ variables;
    Find the set of variables common to all the mini-buckets of variable $X_k$: $\mathbf{S}_k = Scope(q_k^1) \cap \cdots \cap Scope(q_k^t)$;
    $q_k^p$: $F_k^p \leftarrow \prod_{g \in q_k^p} g$;
    Compute the max-marginals of each mini-bucket
    $q_k^p$: $\gamma_{X_k}^p = \max_{Scope(q_k^p) \setminus \mathbf{S}_k} (F_k^p)$;
    Update functions of each mini-bucket
    $F_k^p \leftarrow F_k^p - \gamma_{X_k}^p + \frac{1}{t} \sum_{j=1}^t \gamma_{X_k}^j$;
    Generate messages $h_{X_k \to X_m}^p = \max_{X_k} F_k^p$ and place each in the bucket of highest in the ordering $o$ variable
    $X_m$ in $Scope(q_k^p)$;
return All the buckets and the cost bound from $\mathbf{B}_1$;
```

that the update will help, and in practice we find that the bounds are almost always significantly improved.

Theorem 9.18 Complexity of MBE-MM. *Given a problem with n variables having domain of size k and an i-bound i, MBE-MM is $O(n \cdot Q \cdot k^{i+1})$ time and its space complexity is $O(n \cdot k^i)$, where Q bounds the number of functions sharing the same variable X_i in their scopes.*

CHAPTER 10
Conclusion

We covered the principles of *exact* algorithms in graphical models, organized along the two styles of reasoning: *inference* and *search*. We focused on methods that are applicable to general graphical models, whose functions can come from a variety of frameworks and applications (constraints, Boolean, probabilities, costs etc.). These include, constraint networks and SAT models, Bayesian networks, Markov random fields, Cost networks, and Influence diagrams. Therefore, the primary features that capture structure in a unified way across all these models are graph features. The main graph property is the *induced-width* also known as *treewidth*, but we also showed the relevance of related features such as *height* of pseudo trees, *cycle-cutsets, q-cutsets* and *separator width*. We showed that both inference and search scheme are bounded exponentially by any of these parameters, and some combination of those hint at how we can trade memory for time.

With the exception of constraints, we did not discuss internal function structure as a potential feature. These function-structure features are sometimes addressed as *language* (e.g., Horn clauses, linear functions, convex functions) and can lead to various tractable classes. Other terms used are *context-sensitive* or *context specific independence*. In the constraint literature, tractability based on the language of constraints was investigated thoroughly (see Dechter [50, Chapter 10].) Likewise, focus on language is a central research activity in probabilistic reasoning. An example of a structure exploited in probabilistic graphical models are the sub-modular functions [52].

The next thing on our agenda is to extend the book with a second part focusing on approximation schemes. This obviously is necessary since exact algorithms cannot scale-up to many realistic applications that are complex and quite large and appropriately, current research centered on developing approximation schemes. But, we believe that **in order to have effective approximation algorithms we have to be equipped with the best exact algorithms, first.**

Approximation algorithms can be organized along the dimensions of inference and search as well. Given a general algorithmic architecture (such as Adaptive AND/OR search with caching (AOC(q)), or, alternatively, AO-VEC(q), we can approximate either the inference part or the search part or both, systematically yielding an ensemble of candidates approximation algorithms that can be studied. We can view message-passing and variational algorithms such as generalized belief propagation, the mini-bucket and weighted mini-bucket schemes [43, 87] as approximations that bound inference. We can view Monte Carlo sampling methods, as approximations to search. The hybrid schemes can be used to focus on approximating only those portions of the problem instance that appear non-tractable for exact processing. Namely, for a given problem instances, it can suggest a balance between approximate and exact and the type of approximation that should be utilized.

One should note that approximate reasoning in graphical modeling with any guarantees was shown to be hard as well [32, 114]. Yet, algorithms that generate bounds or anytime schemes that can improve their bounds if allowed more time, and even get to an exact solution when time permits, are highly desirable. Pointers to some literature on approximations can be found in recent Ph.D. theses [80], [22], [69], [100], [94], and in a variety of articles in the field such as (on message-passing variational approaches) [99], [75, 89, 131, 132], [73, 88], and [125]. On Sampling and hybrid of sampling and bounded inference see Bidyuk and Dechter [18], Bidyuk et al. [20] and Gogate and Dechter [65, 66, 67]. On anytime schemes for optimization see Marinescu and Dechter [93], Otten and Dechter [108].

Bibliography

[1] Bar-Yehuda R. A. Becker and D. Geiger. Random algorithms for the loop-cutset problem. In *Uncertainty in AI (UAI)*, pages 81–89, 1999. DOI: 10.1613/jair.638.

[2] M. J. Wainwright and M. I. Jordan. Variational inference in graphical models: The view from the marginal polytope. In *Proceedings of the Annual Allerton congerence on communication control and computing*, volume 41, pages 961–971. Citeseer, 2003. 41

[3] T. Schiex. Arc consistency for soft constraints. *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 411–424, 2000. 41

[4] A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009. DOI: 10.1017/CBO9780511811357.

[5] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000. DOI: 10.1109/18.825794.

[6] D. Allen and A. Darwiche. New advances in inference by recursive conditioning. In *Proc. of the 19th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 2–10, 2003.

[7] K. Kask and R. Dechter. A general scheme for automatic generation of search heuristics from specification dependencies. Artificial Intelligence, 129 (1-2) 91-131, 2001. 25

[8] S. A. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability—a survey. *BIT*, 25:2–23, 1985. DOI: 10.1007/BF01934985.

[9] R. Bar-Yehuda, D. Geiger, J. Naor, and R. M. Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. *SIAM Journal of Computing*, 27(4):942–959, 1998. DOI: 10.1137/S0097539796305109.

[10] R. Bayardo and D. Miranker. A complexity analysis of space-bound learning algorithms for the constraint satisfaction problem. In *Proc. of the 13th National Conference on Artificial Intelligence (AAAI)*, pages 298–304, 1996.

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

[12] A. Becker, R. Bar-Yehuda, and D. Geiger. Randomized algorithms for the loop cutset problem. *Journal of Artificial Intelligence Research (JAIR)*, 12:219–234, 2000. DOI: 10.1613/jair.638.

[13] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983. DOI: 10.1145/2402.322389.

[14] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[15] E. Bensana, M. Lemaitre, and G. Verfaillie. Earth observation satellite management. *Constraints*, 4(3):293–299, 1999. DOI: 10.1023/A:1026488509554.

[16] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, 1972. 18

[17] B. Bidyuk and R. Dechter. On finding w-cutset in Bayesian networks. In *Uncertainty in AI (UAI)*, 2004.

[18] B. Bidyuk and R. Dechter. Cutset sampling for Bayesian networks. *Journal of Artificial Intelligence Research (JAIR)*, 28:1–48, 2007. DOI: 10.1613/jair.2149. 50

[19] G. Hardy, J. Littlewood, and G. Polya. Inequalities. Cambridge Mathematical Library. Cambridge University Press. 1952. 10, 12

[20] B. Bidyuk, R. Dechter, and E. Rollon. Active tuples-based scheme for bounding posterior beliefs. *Journal of Artificial Intelligence Research (JAIR)*, 39:335–371, 2010. DOI: 10.1613/jair.2945. 50

[21] E. Rollon and R. Dechter. New Mini-Bucket Partitioning Heuristics for Bounding the Probability of Evidence Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010. 22

[22] B. Bidyuk. Exploiting graph-cutsets for sampling-based approximations in Bayesian networks. Technical report, Ph.D. thesis, Information and Computer Science, Universiy of California, Irvine, 2006. 50

[23] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the Association of Computing Machinery*, 44(2):165–201, 1997. DOI: 10.1145/256303.256306.

[24] S. Bistarelli. *Semirings for Soft Constraint Solving and Programming (Lecture Notes in Computer Science*. Springer-Verlag, 2004. DOI: 10.1007/b95712.

[25] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *MFCS*, pages 19–36, 1997. DOI: 10.1007/BFb0029946.

[26] C. Borgelt and R. Kruse. *Graphical Models: Methods for Data Analysis and Mining*. Wiley, April 2002.

[27] C. Cannings, E. A. Thompson, and H. H. Skolnick. Probability functions on complex pedigrees. *Advances in Applied Probability*, 10:26–61, 1978. DOI: 10.2307/1426718.

[28] M.-W. Chang, L.-A. Ratinov, and D. Roth. Structured learning with constrained conditional models. *Machine Learning*, 88(3):399–431, 2012. DOI: 10.1007/s10994-012-5296-5.

[29] P. Beam, H. Kautz, Tian Sang, F. Bacchus, and T. Piassi. Combining component caching and clause learning for effective model counting. In *SAT*, 2004.

[30] Z. Collin, R. Dechter, and S. Katz. On the feasibility of distributed constraint satisfaction. In *Proc. of the 12th International Conference of Artificial Intelligence (IJCAI)*, pages 318–324, Sidney, Australia, 1991.

[31] Z. Collin, R. Dechter, and S. Katz. Self-stabilizing distributed constraint satisfaction. *The Chicago Journal of Theoretical Computer Science*, 3(4), special issue on self-stabilization, 1999. DOI: 10.4086/cjtcs.1999.010.

[32] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard (research note). *Artificial Intelligence*, 60:141–153, 1993. DOI: 10.1016/0004-3702(93)90036-B. 1, 50

[33] A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 125(1-2):5–41, 2001. DOI: 10.1016/S0004-3702(00)00069-2.

[34] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association of Computing Machinery*, 7(3), 1960. DOI: 10.1145/321033.321034.

[35] S. de Givry, J. Larrosa, and T. Schiex. Solving max-sat as weighted CSP. In *Principles and Practice of Constraint Programming (CP)*, 2003. DOI: 10.1007/978-3-540-45193-8_25.

[36] 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. DOI: 10.1007/978-3-540-45193-8_25.

[37] R. Dechter and Y. El Fattah. Topological parameters for time-space tradeoff. *Artificial Intelligence*, pages 93–188, 2001. DOI: 10.1016/S0004-3702(00)00050-3.

[38] R. Dechter and R. Mateescu. The impact of and/or search spaces on constraint satisfaction and counting. In *Proc. of Constraint Programming (CP)*, pages 731–736, 2004. DOI: 10.1007/978-3-540-30201-8_56.

[39] R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007. DOI: 10.1016/j.artint.2006.11.003.

[40] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987. DOI: 10.1016/0004-3702(87)90002-6.

[41] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pages 353–366, 1989. DOI: 10.1016/0004-3702(89)90037-4.

[42] R. Dechter and I. Rish. Directional resolution: The Davis–Putnam procedure, revisited. In *Principles of Knowledge Representation and Reasoning (KR)*, pages 134–145, 1994.

[43] R. Dechter and I Rish. Mini-buckets: A general scheme for approximating inference. *Journal of the ACM*, pages 107–153, 2002. 49

[44] R. Dechter and P. van Beek. Local and global relational consistency. *Theoretical Computer Science*, pages 283–308, 1997. DOI: 10.1016/S0304-3975(97)86737-0.

[45] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990. DOI: 10.1016/0004-3702(90)90046-3.

[46] R. Dechter. Constraint networks. *Encyclopedia of Artificial Intelligence*, pages 276–285, 1992. DOI: 10.1002/9780470611821.fmatter.

[47] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proc. 12th Conference on Uncertainty in Artificial Intelligence*, pages 211–219, 1996. DOI: 10.1016/S0004-3702(99)00059-4.

[48] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999. DOI: 10.1016/S0004-3702(99)00059-4. 1

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

[50] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003. 49

[51] R. Dechter. Tractable structures for constraint satisfaction problems. In *Handbook of Constraint Programming, Part I*, chapter 7, pages 209–244, Elsevier, 2006. DOI: 10.1016/S1574-6526(06)80011-8.

[52] S. Dughmi. Submodular functions: Extensions, distributions, and algorithms. A survey. *CoRR*, abs/0912.0322, 2009. 49

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

[54] S. Dalmo F. Bacchus and T. Piassi. Algorithms and complexity results for #sat and Bayesian inference. In *FOCS*, 2003.

[55] S. Dalmo F. Bacchus and T. Piassi. Value elimination: Bayesian inference via backtracking search. In *Uncertainty in AI (UAI)*, 2003.

[56] M. Fishelson and D. Geiger. Exact genetic linkage computations for general pedigrees. *Bioinformatics*, 2002. DOI: 10.1093/bioinformatics/18.suppl_1.S189.

[57] M. Fishelson and D. Geiger. Optimizing exact genetic linkage computations. *RECOMB*, pages 114–121, 2003. DOI: 10.1145/640075.640089.

[58] M. Fishelson, N. Dovgolevsky, and D. Geiger. Maximum likelihood haplotyping for general pedigrees. *Human Heredity*, 2005. DOI: 10.1159/000084736.

[59] E. C. Freuder and M. J. Quinn. The use of lineal spanning trees to represent constraint satisfaction problems. *Technical Report 87-41*, University of New Hampshire, Durham, 1987.

[60] E. C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982. DOI: 10.1145/322290.322292.

[61] E. C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(1):755–761, 1985. DOI: 10.1145/4221.4225.

[62] E. C. Freuder. Partial constraint satisfaction. *Artificial Intelligence*, 50:510–530, 1992. DOI: 10.1016/0004-3702(92)90004-H.

[63] 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.

[64] N. Leone, G. Gottlob, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, pages 243–282, 2000. DOI: 10.1016/S0004-3702(00)00078-3.

[65] V. Gogate and R. Dechter. On combining graph-based variance reduction schemes. In *13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 9:257–264, 2010. 50

[66] V. Gogate and R. Dechter. SampleSearch: Importance sampling in presence of determinism. *Artificial Intelligence*, 175(2):694–729, 2011. DOI: 10.1016/j.artint.2010.10.009. 50

[67] V. Gogate and R. Dechter. Importance sampling-based estimation over and/or search spaces for graphical models. *Artificial Intelligence*, 184-185:38–77, 2012. DOI: 10.1016/j.artint.2012.03.001. 50

[68] V. Gogate, R. Dechter, B. Bidyuk, C. Rindt, and J. Marca. Modeling transportation routines using hybrid dynamic mixed networks. In *UAI*, pages 217–224, 2005.

[69] V. Gogate. Sampling algorithms for probabilistic graphical models with determinism. Technical report, Ph.D. thesis, Information and Computer Science, University of California, Irvine, 2009. 25, 50

[70] 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. DOI: 10.1006/jagm.1995.1009.

[71] R. A. Howard and J. E. Matheson. *Influence Diagrams*. 1984.

[72] A. Ihler, J. Hutchins, and P. Smyth. Learning to detect events with Markov-modulated poisson processes. *ACM Transactions on Knowledge Discovery from Data*, 1(3):13, 2007. DOI: 10.1145/1297332.1297337.

[73] A. T. Ihler, N. Flerova, R. Dechter, and L. Otten. Join-graph based cost-shifting schemes. In *UAI*, pages 397–406, 2012. 50

[74] P. Meseguer, J. Larrosa, and M. Sanchez. Pseudo-tree search with soft constraints. In *European Conference on Artificial Intelligence (ECAI)*, 2002.

[75] W. T. Freeman, J. S. Yedidia, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transaction on Information Theory*, pages 2282–2312, 2005. DOI: 10.1109/TIT.2005.850085. 35, 41, 50

[76] F. V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verlag, New York, 2001.

[77] R. J. Bayardo and R. C. Schrag. Using CSP look-back techniques to solve real world sat instances. In *14th National Conference on Artificial Intelligence (AAAI)*, pages 203–208, 1997.

[78] K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying tree-decompositions for reasoning in graphical models. *Artificial Intelligence*, 166(1-2):165–193, 2005. DOI: 10.1016/j.artint.2005.04.004.

[79] H. Kamisetty, E. P Xing, and C. J. Langmead. Free energy estimates of all-atom protein structures using generalized belief propagation. In *Proc. of the International Conference on Research in Computational Molecular Biology*, pages 366–380, 2007. DOI: 10.1007/978-3-540-71681-5_26.

[80] K. Kask. Approximation algorithms for graphical models. Technical report, Ph.D. thesis, Information and Computer Science, University of California, Irvine, CA, 2001. 50

[81] 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.

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

[83] J. Larrosa. Boosting search with variable-elimination. *CP*, pages 291–305, 2000. DOI: 10.1023/A:1020510611031.

[84] J. Larrosa and R. Dechter. Boosting search with variable elimination in constraint optimization and constraint satisfaction problems. *Constraints*, 8(3):303–326, 2003. DOI: 10.1023/A:1025627211942.

[85] J.-L. Lassez and M. Mahler. On Fourier's algorithm for linear constraints. *Journal of Automated Reasoning*, 9, 1992. DOI: 10.1007/BF00245296.

[86] 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.

[87] Q. Liu and A. T. Ihler. Bounding the partition function using holder's inequality. In *ICML*, pages 849–856, 2011. 19, 49

[88] Q. Liu and A. T. Ihler. Variational algorithms for marginal map. *CoRR*, abs/1302.6584, 2013. 50

[89] T. Jaakola, M. J. Wainwright, and A. S. Willskey. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, pages 2313–2335, 2005. DOI: 10.1109/TIT.2005.850091. 50

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

[91] R. Marinescu and R. Dechter. AND/OR branch-and-bound for graphical models. In *Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 224–229, 2005.

[92] R. Marinescu and R. Dechter. AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16–17):1457–1491, 2009. DOI: 10.1016/j.artint.2009.07.003. 1

[93] R. Marinescu and R. Dechter. Memory intensive AND/OR search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16–17):1492–1524, 2009. DOI: 10.1016/j.artint.2009.07.003. 1, 25, 50

[94] R. Marinescu. AND/OR search strategies for optimization in graphical models. Technical report, Ph.D. thesis, Information and Computer Science, Universiy of California, Irvine, 2007. 50

[95] J. P. Marques-Silva and K. A. Sakalla. Grasp-a search algorithm for propositional satisfiability. *IEEE Transaction on Computers*, pages 506–521, 1999. DOI: 10.1109/12.769433.

[96] R. Mateescu and R. Dechter. The relationship between AND/OR search and variable elimination. In *Proc. of the 21st Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 380–387, 2005.

[97] R. Mateescu and R. Dechter. AND/OR cutset conditioning. In *International Joint Conference on Artificial Intelligence (Ijcai)*, 2005.

[98] R. Mateescu and R. Dechter. A comparison of time-space scheme for graphical models. In *Proc. of the 20th International Joint Conference on Artificial Intelligence*, pages 2346–2352, 2007.

[99] R. Mateescu, K. Kask, V. Gogate, and R. Dechter. Join-graph propagation algorithms. *Journal Artificial Intelligence Research (JAIR)*, 37:279–328, 2010. DOI: 10.1613/jair.2842. 33, 50

[100] R. Mateescu. AND/OR search spaces for graphical models. Technical report, Ph.D. thesis, Information and Computer Science, Universiy of California, Irvine, 2007. DOI: 10.1016/j.artint.2006.11.003. 50

[101] R. McEliece, D. Mackay, and J. Cheng. Turbo decoding as an instance of pearl's "belief propagation" algorithm. 16(2):140–152, February 1998.

[102] L. G. Mitten. Composition principles for the synthesis of optimal multistage processes. *Operations Research*, 12:610–619, 1964. DOI: 10.1287/opre.12.4.610.

[103] P. J. Modi, W. Shena, M. Tambea, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2005. DOI: 10.1016/j.artint.2004.09.003.

[104] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7(66):95–132, 1974. DOI: 10.1016/0020-0255(74)90008-5.

[105] K. P. Murphy. *Machine Learning; A Probabilistic Perspective*. 2012.

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

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

[108] L. Otten and R. Dechter. Anytime AND/OR depth-first search for combinatorial optimization. *AI Communications*, 25(3):211–227, 2012. DOI: 10.3233/AIC-2012-0531. 50

[109] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988. 33, 35

[110] L. Portinale and A. Bobbio. Bayesian networks for dependency analysis: An application to digital control. In *Proc. of the 15th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 551–558, 1999.

[111] 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.

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

[113] I. Rish and R. Dechter. Resolution vs. search; two strategies for sat. *Journal of Automated Reasoning*, 24(1/2):225–275, 2000. DOI: 10.1023/A:1006303512524.

[114] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*. DOI: 10.1016/0004-3702(94)00092-1. 1, 50

[115] 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. DOI: 10.1137/0608024.

[116] T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. *Proc. IJCAI*, pages 542–547, 1999. DOI: 10.1016/S0004-3702(01)00159-X.

[117] L. K. Saul and M. I. Jordan. Learning in Boltzmann trees. *Neural Computation*, 6:1173–1183, 1994. DOI: 10.1162/neco.1994.6.6.1174.

[118] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. 47(1/2/3):7–42, April 2002.

[119] R. Seidel. A new method for solving constraint satisfaction problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 338–342, 1981.

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

[121] P. P. Shenoy. Valuation-based systems for Bayesian decision analysis. *Operations Research*, 40:463–484, 1992. DOI: 10.1287/opre.40.3.463.

[122] P. P. Shenoy. Binary join trees for computing marginals in the Shenoy–Shafer architecture. *International Journal of Approximate Reasoning*, pages 239–263, 1997. DOI: 10.1016/S0888-613X(97)89135-9.

[123] K. Shoiket and D. Geiger. A proctical algorithm for finding optimal triangulations. In *14th National Conference on Artificial Intelligence (AAAI)*, pages 185–190, 1997.

[124] J. Sivic, B. Russell, A. Efros, A. Zisserman, and W. Freeman. Discovering object categories in image collections. In *Proc. International Conference on Computer Vision*, 2005.

[125] D. Sontag, T. Meltzer, A. Globerson, T. Jaakkola, and Y. Weiss. Tightening LP relaxations for map using message passing. In *UAI*, pages 503–510, 2008. 50

[126] 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. DOI: 10.1137/0213035.

[127] C. Terrioux and P. Jegou. Bounded backtracking for the valued constraint satisfaction problems. In *Constraint Programming (CP)*, pages 709–723, 2003.

[128] C. Terrioux and P. Jegou. Hybrid backtracking bounded by tree-decomposition of constraint networks. In *Artificial Intelligence*, 2003. DOI: 10.1016/S0004-3702(02)00400-9.

[129] P. Thébault, S. de Givry, T. Schiex, and C. Gaspin. Combining constraint processing and pattern matching to describe and locate structured motifs in genomic sequences. In *5th IJCAI Workshop on Modelling and Solving Problems with Constraints*, 2005.

[130] P. Beam, H. Kautz, Tian Sang, F. Bacchus, and T. Piassi. Combining component caching and clause learning for effective model counting. In *SAT*, 2004.

[131] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008. DOI: 10.1561/2200000001. 50

[132] M. J. Wainwright, T. Jaakkola, and A. S. Willsky. Tree-based reparameterization framework for analysis of sum-product and related algorithms. *IEEE Transactions on Information Theory*, 49(5):1120–1146, 2003. DOI: 10.1109/TIT.2003.810642. 50

[133] Y. Weiss and J. Pearl. Belief propagation: Technical perspective. *Communications ACM*, 53(10):94, 2010. DOI: 10.1145/1831407.1831430.

[134] A. Willsky. Multiresolution Markov models for signal and image processing. 90(8):1396–1458, August 2002.

[135] C. Yanover and Y. Weiss. Approximate inference and protein folding. In *Proc. of Neural Information Processing Systems Conference*, pages 84–86, 2002.

[136] N. L. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research (JAIR)*, 1996. DOI: 10.1613/jair.305.

Author's Biography

RINA DECHTER

Rina Dechter's research centers on computational aspects of automated reasoning and knowledge representation including search, constraint processing, and probabilistic reasoning. She is a Chancellor's Professor of Computer Science at the University of California, Irvine. She holds a Ph.D. from UCLA, an M.S. degree in applied mathematics from the Weizmann Institute, and a B.S. in mathematics and statistics from the Hebrew University in Jerusalem. She is the author of *Constraint Processing* published by Morgan Kaufmann (2003), and of *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms* published by Morgan and Claypool (2013). She has co-authored close to 200 research papers and has served on the editorial boards of: *Artificial Intelligence*, the *Constraint Journal*, *Journal of Artificial Intelligence Research (JAIR)*, and *Journal of Machine Learning Research (JMLR)*. She is a Fellow of the American Association of Artificial Intelligence since 1994, was a Radcliffe Fellow during 2005–2006, received the 2007 Association of Constraint Programming (ACP) Research Excellence Award, and became an ACM Fellow in 2013. She was a Co-Editor-in-Chief of *Artificial Intelligence* from 2011 to 2018 and is the conference chair-elect for IJCAI-2022.