Chapter 8

Bounding Inference

Up to now we focused almost exclusively on exact algorithms for processing graphical models and we emphasized the two styles of inference (exemplified by variable elimination schemes) and search, or conditioning, exemplified by AND/OR search or backtracking search (for constraint networks). We also showed that hybrids of search and inference are effective and can be used to trade space for time.

Clearly, due to the hardness of the tasks we address, some networks cannot be processed exactly because their structure is not sparse enough; its treewidth is too high, and the functions themselves do not posses any property that can be exploited. In such cases approximation algorithms are the only choice. Approximation algorithms can also be designed as either approximating an inference scheme, or as approximating search. Bounded inference algorithms, on which this chapter focuses, approximate inference, while sampling scheme can be viewed as approximating search.

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

It was shown that approximation scheme within given relative error bounds is NP-hard [57, 67]. Nevertheless there are approximation strategies that work well in practice. One alternative for dealing with these bleak fact is to develop *anytime algorithms*. These algorithms can be interrupted at any time producing the best solution found thus far. Other methodology is to exploit the special structure of the problem or to aim at providing

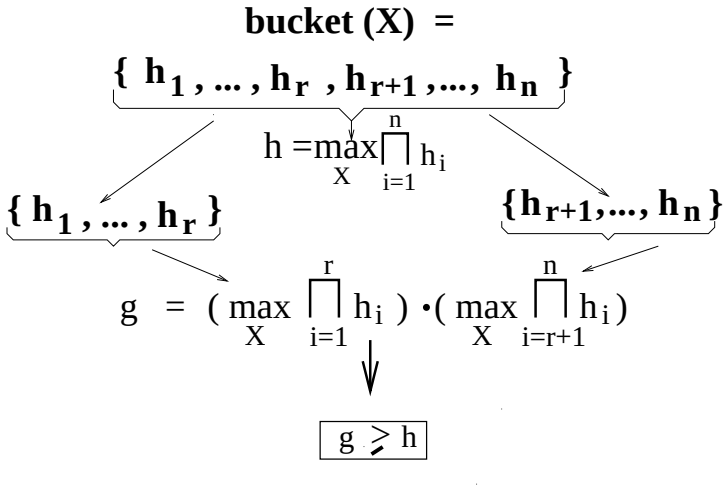


Figure 8.1: The idea of mini-bucket approximation.

some guarantee in the form of upper and lower bounds.

The class of *mini-bucket* approximation algorithms imports some ideas of *local inference* from constraint networks to probabilistic reasoning and combinatorial optimization. As we showed (Chapter **??**) the bucket-elimination algorithm is a unifying algorithmic scheme that generalizes non-serial dynamic programming to enable complex problem-solving and reasoning across a variety of frameworks. As was shown in Chapters 3 and 4, among the algorithms that can be expressed as bucket-elimination are *directional-resolution* for propositional satisfiability *adaptive-consistency* for constraint satisfaction, *Fourier* and *Gaussian elimination* for linear inequalities, *dynamic-programming* for combinatorial optimization as well as many algorithms for probabilistic inference [17].

In the following sections we will introduce the mini-bucket elimination scheme.

8.1 Mini-bucket approximation for MPE

We will introduce the idea of mini-bucket approximation using the combinatorial optimization task of finding the most probable explanation, MPE.

Consider the bucket-elimination algorithm *BE-mpe*. Since the complexity of processing a bucket depends on the number of arguments (arity) of the functions being recorded, we should consider approximating these functions by a collection of smaller-arity functions.

Let $h_1, ..., h_t$ be the functions in the bucket of X_p, and let $S_1, ..., S_t$ be their scopes. When *BE-mpe* processes bucket(X_p), the function $h^p = max_{X_p} \Pi_{i=1}^t h_i$ is computed. A simple approximation idea is to compute an upper bound on h^p by "migrating" the maximization inside the multiplication. Since, in general, for any two non-negative functions $Z(x)$ and $Y(x)$, $\max_x Z(x) \cdot Y(x) \leq \max_x Z(x) \cdot \max_x Y(x)$, this approximation will compute an upper bound on h^p. For example, the function $g^p = \Pi_{i=1}^t \max_{X_p} h_i$, is an upper bound on h^p. Procedurally it implies that the elimination operation of maximization is applied separately to each function, requiring less computation (when h_i's scopes include additional variables).

The idea is demonstrated in Figure 8.1, where the bucket of variable X having n functions is split into two mini-buckets of size r and $(n - r)$, $r \leq n$, and it can be generalized to any partitioning of a set of functions $h_1, ..., h_t$ into subsets called *mini-buckets*. Let $Q = \{Q_1, ..., Q_r\}$ be a partitioning into mini-buckets of the functions $h_1, ..., h_t$ in X_p's bucket, where the mini-bucket Q_l contains the functions $h_{l_1}, ..., h_{l_r}$. The complete algorithm *BE-mpe* computes $h^p = \max_{X_p} \Pi_{i=1}^t h_i$, which can be rewritten as $h^p = \max_{X_p} \Pi_{l=1}^r \Pi_{h \in Q_l} h$. By migrating maximization into each mini-bucket we can compute: $g_Q^p = \Pi_{l=1}^r \max_{X_p} \Pi_{h \in Q_l} h$. The new functions $\max_{X_p} \Pi_{h \in Q_l} h$ can now be placed separately into the bucket of the highest-variable in their scope and the algorithm proceeds with the next variable. Functions without arguments (i.e., constants) are placed in the lowest bucket. The reader should be convinced that the maximized product generated in the first bucket is an upper bound on the MPE value. Finally, A lower bound can also be computed as the probability of a (suboptimal) assignment found in the forward phase of the algorithm.

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

Clearly we would want the mini-buckets to be as small as possible yet we wish the scheme to be as accurate as possible. There are certain obvious restrictions on possible partitions into mini-buckets which will exclude some obvious unwarranted choice. Clearly, we would want to have a small number of mini-buckets which also means that we want to

Algorithm mbe-mpe(i,m)
Input: A belief network $BN = (G, P)$, an ordering o, evidence $\bar{e}$.
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 $L = P(\bar{x}^a)$.

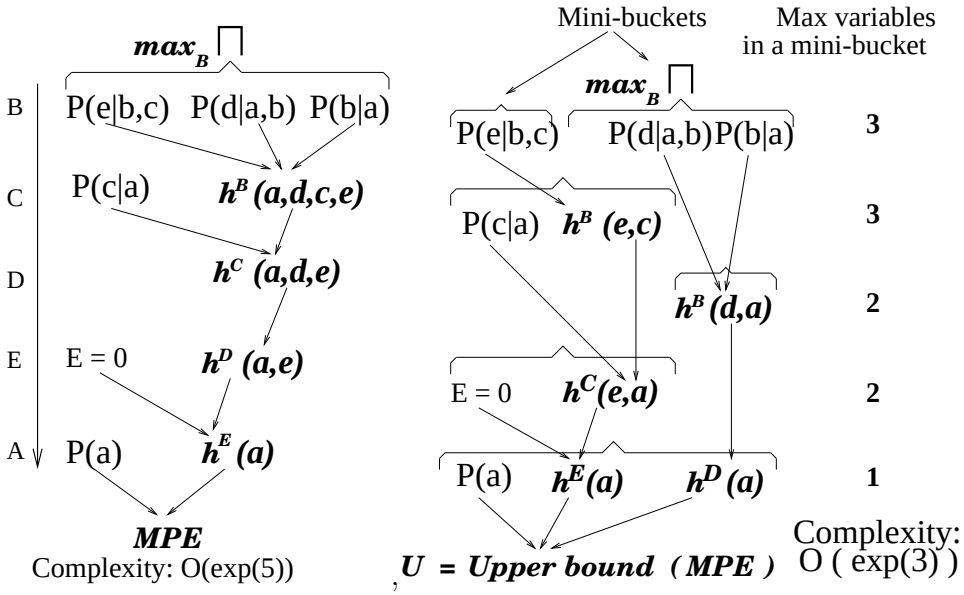
1. **Initialize:** Partition $P = \{P_1, ..., P_n\}$ into buckets $bucket_1$, ..., $bucket_n$, where $bucket_p$ contains all CPTs $h_1, h_2, ..., h_t$ whose highest-index variable is X_p.
2. **Backward: for** $p = n$ to 2 **do**
   - **If** X_p is observed ($X_p = a$), assign $X_p = a$ in each h_j and put the result in its highest-variable bucket (put constants in $bucket_1$).
   - **Else** for $h_1, h_2, ..., h_t$ in $bucket_p$ do
     Generate an (i, m)-mini-bucket-partitioning, $Q' = \{Q_1, ..., Q_r\}$.
     for each $Q_l \in Q'$ containing $h_{l_1}, ...h_{l_t}$, **do**
       compute $h^l = max_{X_p} \Pi_{j=1}^t h_{l_j}$ and place it in the bucket of the highest-index variable in $U_l \leftarrow \bigcup_{j=1}^t S_{l_j} - \{X_p\}$, where S_{l_j} is the scope of h_{l_j} (put constants in $bucket_1$).
3. **Forward: for** $p = 1$ to n, given $x_1^a, ..., x_{p-1}^a$, **do**
   assign a value x_p^a to X_p that maximizes the product of all functions in $bucket_p$.
4. **Return** the assignment $\bar{x}^a = (x_1^a, ..., x_n^a)$, a lower bound $L = P(\bar{x}^a)$, and an upper bound $U = max_{x_1} \prod_{h_j \in bucket_1} h^j$ on the $MPE = \max_{\bar{x}} P(\bar{x}, \bar{e})$.

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

have as many functions as possible in each mini-bucket. Clearly therefore, if one function scope subsumed another, we would wish that to share the same mini-bucket.

Definition 8.1.1 ((i,m)-partitioning) *A partitioning of $h_1, ..., h_t$ is* canonical *if any function f whose scope is subsumed by the scope of another function, is placed into a bucket containing one of those subsuming functions. A partitioning Q into mini-buckets is an (i, m)-partitioning if and only if (1) it is canonical, (2) at most m non-subsumed functions are included in each mini-bucket, (3) the total number of variables in a mini-bucket does not exceed i, and (4) the partitioning is* refinement-maximal*, namely, there is no other (i, m)-partitioning that it refines.*

The parameters i (number of variables) and m (number of functions allowed per mini-bucket) are not independent, and some combinations of i and m do not allow an (i,m)-partitioning. However, it is easy to see that If the bound i on the number of variables in



(a) A trace of *elim-mpe* (b) A trace of *mbe-mpe(3,2)*.

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

a mini-bucket is not smaller than the maximum family 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 a richer set of partitioning schemes than using i or m alone. Since *mbe-mpe(i,m)* computes an upper bound in each bucket it yields an overall upper bound on the resulting MPE.

Theorem 8.1.2 (mbe-mpe properties) *Algorithm* mbe-mpe*(i, m) computes an upper and lower bounds on the MPE.*

We will prove the theorem later (section 8.5) in a more general setting, common to all mini-bucket elimination algorithms.

In general, as m and i increase, we get more accurate approximations. Note, however, a monotonic increase in accuracy as a function of i can be guaranteed only for refinements of a given partitioning as we discuss next.

Definition 8.1.3 *Given two partitionings Q' and Q'' over the same set of elements, Q' is a refinement of Q'' if and only if for every set $A \in Q'$ there exists a set $B \in Q''$ such that $A \subseteq B$.*

It is easy to see that:

Proposition 8.1.4 *If $Q^{''}$ is a refinement of $Q^{'}$ in bucket$_p$, then $h^p \leq g^p_{Q'} \leq g^p_{Q''}$.*

Proof: Clearly for any partitioning Q we have $h^p \leq g^p_Q$.

By definition, given a refinement $Q'' = \{Q''_1, ..., Q''_k\}$ of a partitioning $Q' = \{Q'_1, ..., Q'_m\}$, each mini-bucket $i \in \{1, ..., k\}$ of Q'' belongs to some mini-bucket $j \in \{1, ..., m\}$ of Q'. In other words, each mini-bucket j of Q' is further partitioned into the corresponding mini-buckets of Q'', $Q'_j = \{Q''_{j_1}, ..., Q''_{j_l}\}$. Therefore,

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

■

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

On the other hand, since bucket(B) includes five variables, *mbe-mpe(3,2)* splits it into two mini-buckets $\{P(e|b, c)\}$ and $\{P(d|a, b), P(b|a)\}$, each containing no more than 3 variables, as shown in Figure 8.3b (the (3,2)-partitioning can be selected arbitrarily). The new functions $h^B(e, c)$ and $h^B(d, a)$ are generated in different mini-buckets and are placed independently in 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 is computed by maximizing over A the product of functions in A's bucket: $U = max_a P(a)h^E(a)h^D(a)$. Once all the buckets are processed, a suboptimal MPE tuple is computed by assigning a value to each variable that maximizes the product of functions in the corresponding bucket, as if exact BE-mpe is applied. 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, algorithm *mbe-mpe(i,m)* computes an interval $[L, U]$ containing the highest MPE value where U is the upper bound computed by the backward phase and L is the probability of the returned assignment.

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

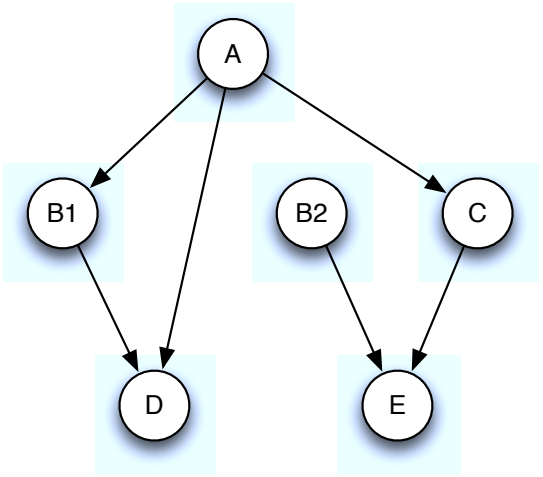
$$\frac{L}{P(\bar{e})} \leq M \leq \frac{U}{P(\bar{e})}$$

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

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 of 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 variables, each corresponding to a single mini-bucket. In the resulting relaxed problem, each function is associated with the copy of the variable corresponding to its mini-bucket in the original problem. For example, the Mini-Bucket trace in Figure 8.3b, corresponds to solving exactly by full bucket-elimination the network of the problem in Figure 8.4. Variable B is replaced by two 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 is equal to the upper bound we know that we found the optimal solution. Alternatively, if we use the node duplication explicitly, whenever the optimal solution assign the same value to duplicated variables, we would also know that the solution is optimal, even if we did not have an upper-bound to compare with.

As we will see next, approximating conditional probabilities using bounds on joint probabilities may be more problematic for belief updating.



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

(b)

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

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* finds $P(x_1, \bar{e})$ and then computes $P(x_1|\bar{e}) = \alpha P(x_1, \bar{e})$ where α is the normalization constant (see Figure 4.4).

The mini-bucket idea used for approximating MPE can be applied to belief updating. Let $Q\prime = \{Q_1, ..., Q_r\}$ be a partitioning of the functions $h_1, ...h_t$ (defined over scopes $S_1, ..., S_t$, respectively) in X_p's bucket. Algorithm *BE-bel* computes $h^p : U_p \rightarrow \Re$, where $h^p = \sum_{X_p} \Pi_{i=1}^t h_i$, and $U_p = \cup_i S_i - \{X_p\}$. The function h_p can be rewritten as $h^p = \sum_{X_p} \Pi_{l=1}^r \Pi_{l_i|i=1,..,j_l} h_{l_i}$. If we follow the MPE approximation precisely and migrate the summation operator into each mini-bucket, we will get $f^p_{Q\prime} = \Pi_{l=1}^r \sum_{X_p} \Pi_{l_i} h_{l_i}$. This, however, is an unnecessarily large upper bound of h^p in which each $\Pi_{l_i} h_{l_i}$ that is a function of X_p is bounded by $\sum_{X_p} \Pi_{l_i} h_{l_i}$, a constant relative to X_p. Instead, we rewrite $h^p = \sum_{X_p} (\Pi_{1_i} h_{1_i}) \cdot (\Pi_{l=2}^r \Pi_{l_i} h_{l_i})$. Subsequently, instead of bounding a function of X by its sum over X, we can bound $(i > 1)$, by its maximum over X, yielding $g^p_{Q\prime} = (\sum_{X_p} \Pi_{1_i} h_{1_i}) \cdot (\Pi_{l=2}^r \max_{X_p} \Pi_{l_i} h_{l_i})$. Therefore, an upper bound g^p of h^p can be obtained by processing

Algorithm mbe-bel-max(i,m)
Input: A belief network $BN = (G, P)$, an ordering o, and evidence $\bar{e}$.
Output: an upper bound on $P(x_1, \bar{e})$ and an upper bound on $P(e)$.
1. **Initialize:** Partition $P = \{P_1, ..., P_n\}$ into buckets $bucket_1$, ..., $bucket_n$, where $bucket_k$ contains all CPTs $h_1, h_2, ..., h_t$ whose highest-index variable is X_k.
2. **Backward:** for $k = n$ to 2 do
   - **If** X_p is observed ($X_k = a$), assign $X_k \leftarrow a$ in each h_j and put the result in the highest-variable bucket of its scope (put constants in $bucket_1$).
   - **Else** for $h_1, h_2, ..., h_t$ in $bucket_k$ do
     Generate an (i, m)-mini-bucket-partitioning, $Q' = \{Q_1, ..., Q_r\}$.
     For each $Q_l \in Q'$, containing $h_{l_1}, ...h_{l_t}$, do
       If $l = 1$ compute $h^l = \sum_{X_k} \Pi_{j=1}^t h_{1_j}$
       Else compute $h^l = max_{X_k} \Pi_{j=1}^t h_{l_j}$
       Add h^l to the bucket of the highest-index variable in $U_l \leftarrow \bigcup_{j=1}^t S_{l_j} - \{X_k\}$, (put constant functions in $bucket_1$).
3. **Return** $P'(\bar{x_1}, e) <-- $ the product of functions in the bucket of X_1, which is an upper bound on $P(x_1, \bar{e})$.
$P'(e) <-- \sum_{x_1} P'(\bar{x_1}, e)$, which is an upper bound on probability of evidence.

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

one of X_p's mini-buckets by summation and the rest by maximization.

A lower bound on the belief, or its mean value, can be obtained in a similar way. Algorithm *mbe-bel-max(i,m)* that uses the *max* elimination operator is described in Figure 8.5. Algorithms *mbe-bel-min* and *mbe-bel-mean* can be obtained by replacing the operator *max* by *min* and by *mean*, respectively.

Notice that the node duplication semantics of MNE implies summation for each of the mini-buckets.

Theorem 8.2.1 *Given a Bayesian network with evidence e, algorithm MBE-bel-max(i,m) computes an upper bound on $P(X_1, e)$ and $P(e)$ in time and space $O(r \cdot k^i)$, when r is the number of functions and k bound the domain size.*

We will have the same relationships between partitionings and their refinements as for the mpe case.

Proposition 8.2.2 *For every partitioning Q and its variable X_p, $h^p \leq g^p_Q \leq f^p_Q$, where in f each mini-bucket is processed by summation and in g, one is processed by summation and the rest by maximization. Also, if Q'' is a refinement partitioning of Q', then $h^p \leq g^p_{Q'} \leq g^p_{Q''}$.*

8.2.1 Normalization

Note that *aprox-bel-max* computes an upper bound on $P(x_1, \bar{e})$ but not on $P(x_1|\bar{e})$. If an exact value of $P(\bar{e})$ is not available, deriving a bound on $P(x_1|\bar{e})$ from a bound on $P(x_1, \bar{e})$ is not easy, because $\frac{g(x_1)}{\sum_{x_1} g(x_1)}$, where $g(x)$ is the upper bound on $P(x_1, \bar{e})$, is not necessarily an upper bound on $P(x_1|\bar{e})$. As noted we can derive a lower bound, f, on $P(\bar{e})$ using *mbe-bel-min* and then compute $\frac{g(x_1)}{f}$ as an upper bound on $P(x_1|\bar{e})$. This however is likely to generate weak bounds due to compounded error.

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

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

Therefore, although $P(\bar{e})$ is not known, the ratio of upper to lower bounds remains the same. Yet, the difference between the upper and the lower bounds can grow substantially,

especially in cases of rare evidence. Note that if $P(\bar{e}) \leq U_i$, we get $\frac{L_i}{P(\bar{e})} \leq P(X_1|\bar{e}) \leq 1$, yielding a trivial upper bound. Finally, note that no guarantee is given for $g_{mean}(x_i)$, and therefore, the approximation of $\frac{g_{mean}(x_i)}{\sum_{x_1} g_{mean}(x_1)}$ can be below or above the exact value. Interestingly, the computation of $\frac{g_{mean}(X_1=x_i)}{\sum_{x_1} g_{mean}(x_1)}$ is achieved when processing all mini-buckets by summations, and subsequently normalizing. Namely, this is what is obtained when we transform the original network by node duplication and apply exact BE-bel to the simplified network.

8.3 Mini-bucket elimination for MAP

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

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

$$\bar{a}_k^{map} = \arg\max_{\bar{a}_k} P(\bar{a}_k|\bar{e}) = \arg\max_{\bar{a}_k} \frac{\sum_{\bar{x}_{k+1}^n} \Pi_{i=1}^n P(x_i, \bar{e}|x_{pa_i}^-)}{P(\bar{e})} \tag{8.1}$$

where $\bar{x} = (a_1, ..., a_k, x_{k+1}, ..., x_n)$ denotes an assignment to all variables, while $\bar{a}_k = (a_1, ..., a_k)$ and $\bar{x}_{k+1}^n = (x_{k+1}, ..., x_n)$ denote assignments to the hypothesis and non-hypothesis variables, respectively. Since $P(\bar{e})$ is a normalization constant, the maximum of $P(\bar{a}_k|\bar{e})$ is achieved at the same point as the maximum of $P(\bar{a}_k, \bar{e})$ (as before, we have $P(\bar{a}_k|\bar{e}) = \frac{P(\bar{a}_k,\bar{e})}{P(\bar{e})}$.) and therefore we define $MAP = P(\bar{a}_k, \bar{e})$. We will derive an approximation to this quantity which is easier than approximating $P(\bar{a}_k|\bar{e})$.

The bucket-elimination algorithm for finding the exact MAP, *BE-map*, assumes only orderings in which the hypothesized variables appear first and thus are processed last by the algorithm, as discussed earlier. This restriction makes the task more difficult because it implies higher induced widths. The algorithm has the usual backward phase. Its forward phase however is relative to the hypothesis variables only. The mini-bucket scheme for map is a straightforward extension of the algorithms *mbe-mpe* and *mbe-bel-max*. We partition each bucket into mini-buckets as before. If the bucket's variable can be eliminated by summation, we apply the rule we have in *mbe-bel-max* in which one

Algorithm mbe-map(i,m)
Input: A belief network $BN = (G, P)$, a subset of variables $A = \{A_1, ..., A_k\}$, an ordering of the variables, o, in which the A's appear first, and evidence $\bar{e}$.
Output: An upper bound U on the MAP and a suboptimal solution $A = \bar{a}^a_k$.
1. **Initialize:** Partition $P = \{P_1, ..., P_n\}$ into buckets $bucket_1, \ldots, bucket_n$ where $bucket_P$ contains all CPTs, $h_1, ..., h_t$ whose highest index variable is X_p.
2. **Backward:** for $p = n$ to 1 do
   - **If** X_p is observed ($X_p = a$), assign $X_p = a$ in each h_i and put the result in its highest-variable bucket (put constants in $bucket_1$).
   - **Else** for $h_1, h_2, ..., h_j$ in $bucket_p$ do
     Generate an (i, m)-partitioning, $Q^{'}$ of the matrices h_i into mini-buckets $Q_1, ..., Q_r$.
     - **If** $X_P \notin A$ /* not a hypothesis variable */
       for each $Q_l \in Q^{'}$, containing $h_{l_1}, ...h_{l_t}$, do
       If $l = 1$, compute $h^l = \sum_{X_p} \Pi_{i=1}^t h_{1_i}$
       Else compute $h^l = max_{X_p} \Pi_{i=1}^t h_{l_i}$
       Add h^l to the bucket of the highest-index variable in $U_l \leftarrow \bigcup_{i=1}^t S_{l_i} - \{X_p\}$, (put constants in $bucket_1$).
     - **Else** ($X_p \in A$) /* a hypothesis variable */
       for each $Q_l \in Q^{'}$ containing $h_{l_1}, ...h_{l_t}$ compute $h^l = max_{X_p} \Pi_{i=1}^t h_{l_i}$ and place it in the bucket of the highest-index variable in $U_l \leftarrow \bigcup_{i=1}^t S_{l_i} - \{X_p\}$, (put constants in $bucket_1$).
3. **Forward: for** $p = 1$ to k, given $A_1 = a^a_1, ..., A_{p-1} = a^a_{p-1}$, assign a value a^a_p to A_p that maximizes the product of all functions in $bucket_p$.
4. **Return** An upper bound $U = max_{a_1} \prod_{h_i \in bucket_1} h_i$ on MAP, computed in the first bucket. and the assignment $\bar{a}^a_k = (a^a_1, ..., a^a_k)$.

Figure 8.6: Algorithm *mbe-map(i,m)*.

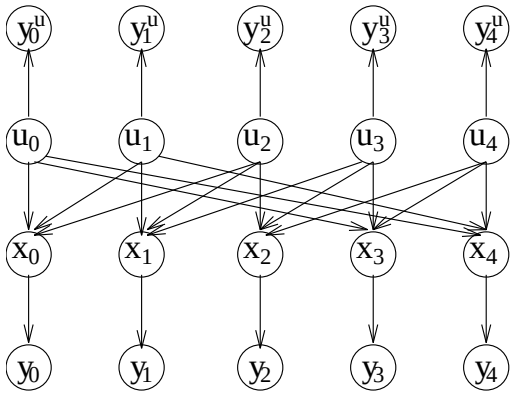


Figure 8.7: Belief network for a linear block code.

mini-bucket is approximated by summation and the rest by maximization. When the algorithm reaches the hypothesis buckets, their processing is identical to that of *mbe-mpe*. Algorithm *mbe-map(i,m)* is described in Figure 8.6.

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

Exercise: Show how we can obtain both a lower bound and an upper-bound on the map query.

Example 8.3.1 Consider a belief network which describes the decoding of a *linear block code*, shown in Figure 8.7. In this network, U_i are *information bits* and X_j are *code bits*, which are functionally dependent on U_i. The vector (U, X), called the channel input, is transmitted through a noisy channel which adds Gaussian noise and results in the channel output vector $Y = (Y^u, Y^x)$. The decoding task is to assess the most likely values for the U's given the observed values $Y = (\bar{y}^u, \bar{y}^x)$, 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)* of the first top five buckets by summation and the rest by maximization, results in the following mini-bucket partitionings and function generation:
$bucket(X_0) = \{P(y_0^x|X_0), P(X_0|U_0, U_1, U_2)\},$
$bucket(X_1) = \{P(y_1^x|X_1), P(X_1|U_1, U_2, U_3)\},$
$bucket(X_2) = \{P(y_2^x|X_2), P(X_2|U_2, U_3, U_4)\},$
$bucket(X_3) = \{P(y_3^x|X_3), P(X_3|U_3, U_4, U_0)\},$
$bucket(X_4) = \{P(y_4^x|X_4), P(X_4|U_4, U_0, U_1)\},$
$bucket(U_0) = \{P(U_0), P(y_0^u|U_0), h^{X_0}(U_0, U_1, U_2)\}, \{h^{X_3}(U_3, U_4, U_0)\}, \{h^{X_4}(U_4, U_0, U_1)\},$
$bucket(U_1) = \{P(U_1), P(y_1^u|U_1), h^{X_1}(U_1, U_2, U_3), h^{U_0}(U_1, U_2)\}, \{h^{U_0}(U_4, U_1)\},$
$bucket(U_2) = \{P(U_2), P(y_2^u|U_2), h^{X_2}(U_2, U_3, U_4), h^{U_1}(U_2, U_3)\},$
$bucket(U_3) = \{P(U_3), P(y_3^u|U_3), h^{U_0}(U_3, U_4), h^{U_1}(U_3, U_4), h^{U_2}(U_3, U_4)\},$
$bucket(U_4) = \{P(U_4), P(y_4^u|U_4), h^{U_1}(U_4), h^{U_3}(U_4)\}.$
The first five buckets are not partitioned at all and are processed as full buckets, since in this case a full bucket is a (4,1)-partitioning. This processing generates five new functions, three are placed in bucket U_0, one in bucket U_1 and one in bucket U_2. Then bucket U_0 is partitioned into three mini-buckets processed by maximization, creating two functions placed in bucket U_1 and one function placed in bucket U_3. Bucket U_1 is partitioned into two mini-buckets, generating functions placed in bucket U_2 and bucket U_3. Subsequent buckets are processed as full buckets. Note that the scope of recorded functions is bounded by 3.

In the bucket of U_4 we get an upper bound U satisfying $U \geq MAP = P(U, \bar{y}^u, \bar{y}^x)$ where $\bar{y}^u$ and $, \bar{y}^x$ are the observed outputs for the U's and the X's bits transmitted. In order to bound $P(U|\bar{e})$, where $\bar{e} = (\bar{y}^u, \bar{y}^x)$, we need $P(\bar{e})$ which is not available.

Yet, again, in most cases we are interested in the ratio $P(U = \bar{u_1}|\bar{e})/P(U = \bar{u_2}|\bar{e})$ for competing hypotheses $U = \bar{u_1}$ and $U = \bar{u_2}$ rather than in the absolute values. Since $P(U|\bar{e}) = P(U, \bar{e})/P(\bar{e})$ and the probability of the evidence is just a constant factor independent of U, the ratio is equal to $P(U_1, \bar{e})/P(U_2, \bar{e})$. □

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?

8.4 Mini-buckets for discrete optimization

The mini-bucket principle can also be applied to deterministic discrete optimization problems which can be defined over *cost networks*, yielding approximation to dynamic programming for discrete optimization [9]. In fact, 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 sake we present the algorithm explicitly within the framework of cost networks.

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

Algorithm *mbe-opt* is described for the sake of completeness in Figure 8.8. 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. We leave it as an exercise.

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

Figure 8.8: Algorithm *mbe-opt(i,m)*.

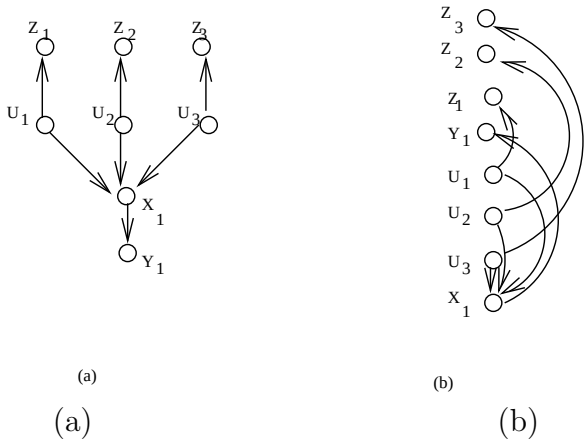
8.5 Complexity and tractability

8.5.1 The case of low induced width

We denote by *mini-bucket-elimination(i,m)*, or simply *mbe(i,m)*, a generic mini-bucket scheme with parameters i and m, without specifying the particular task it solves, which can be either one of probabilistic inference tasks defined above or a general constrained optimization problem.

It is easy to derive *mbe(i,m)* complexity as the bucket-elimination complexity of the relaxed problem generated by variable/node duplication. Since node duplication can generate at most r additional variables but will leave the number of functions fixed at r, and since the resulting problem has induced-width bounded by i, BE complexity on such a problem is time complexity $O(r \cdot k^i)$.

Theorem 8.5.1 *Algorithm* mbe(i,m) *takes* $O(r \cdot exp(i))$ *time and space, where* r *is the*



(a) (b)

Figure 8.9: (a) A polytree and (b) a legal ordering, assuming that nodes Z_1, Z_2, Z_3 and Y_1 are observed.

number of input functions[1]*. For* $m = 1$, mbe(i,1) *is linear. It is time and space linear and is bounded by* $O(r \cdot exp(|S|))$*, where* $|S|$ *is the maximum scope of any input function,* $|S| \leq i \leq n$.

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.

Theorem 8.5.2 *Given an ordering of the variables, o, algorithm* mbe(i, n) *applied along o is complete for networks having* $w_o^* \leq i$.

It is interesting to note that when $m = 1$ the algorithm *mbe(i,m)* is exact for any acyclic network, and in particular for for polytrees, if applied along some *appropriate* orderings. Such suitable orderings are determined be consulting a rooted join-tree of the acyclic network (we know that such exists from the definition of an acyclic network as discussed in Chapter 6). We can now order the variables from last to first by selecting and removing a leaf function from the rooted join-tree and placing all its variables that were not ordered yet, consecutively, next in the ordering. With such orderings, each bucket would contain at most one non-subsumed function.

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

Theorem 8.5.3 *Given an acyclic network (e.g., a polytree), there exists an ordering o such that algorithm* mbe(n,1) *is exact and is time and space* $O(n \cdot exp(|S|))$*, where* $|S|$ *is the largest scope size of any input function.*

Example 8.5.4 Consider an ordering $o = (X_1, U_3, U_2, U_1, Y_1, Z_1, Z_2, Z_3)$ of the polytree in Figure 8.9a, where the last four variables $Y_1,\ Z_1,\ Z_2,\ Z_3$ in the ordering are observed. Once the last four buckets were already processed as observation buckets, we get (observed values shown in low-case):
$bucket(U_1) = P(U_1), P(X_1|U_1, U_2, U_3), P(z_1|U_1),$
$bucket(U_2) = P(U_2), P(z_2|U_2),$
$bucket(U_3) = P(U_3), P(z_3|U_3)$
$bucket(X_1) = P(y_1|X_1).$

□

Not coincidentally, on polytrees, *mbe(n,1)* is similar to one pass of Pearl's well-known propagation algorithm.

8.6 Using the Mini-bucket scheme

8.6.1 Anytime inference

An important property of the mini-bucket scheme is that it provides an adjustable trade-off between accuracy of solution and computational complexity. Both the accuracy and the complexity increase with increasing 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 the *anytime* framework.

Anytime algorithms can be interrupted at any time producing the best solution found thus far. As more time is available, better solutions will be generated. Clearly, an iterative application of such schemes with less restrictions on the amount of information they use, results in an anytime inference algorithms that eventually become exact, if sufficient computational resources are available.

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

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

Figure 8.10: Algorithm *anytime-mpe(ϵ)*.

until the computational resources are exhausted. This anytime version is particularly attractive for an optimization task such as the most probable explanation (mpe). The anytime algorithm *anytime-mpe(ϵ)* is presented in Figure 8.10. The parameter ϵ is the desired accuracy level. The algorithm uses initial parameter settings, i_0 and m_0, and increments i_{step} and m_{step}. Starting with $i = i_0$ and $m = m_0$, *mbe-mpe(i,m)* computes a suboptimal MPE solution and the corresponding lower bound L, and an upper bound (U) for increasing values of i and m. The algorithm terminates when either $1 \leq U/L \leq 1 + \epsilon$, or when the computational resources are exhausted, returning the largest lower bound and the smallest upper bound found so far, as well as the current best suboptimal solution. Note that the algorithm is complete when $\epsilon = 0$.

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

The above scheme can be refined and improved to save redundant computation in the repeated mini-bucket runs and also in controlling the partitioning from one run to the next in a manner that will guarantee improvements (by combining some mini-buckets, for

example). You may want to consider the following exercise: design an anytime scheme that allows computation sharing between subsequent mbe processing.

Another orthogonal anytime extension of the mini-bucket, is to embed it within a complete anytime heuristic search algorithm such as *branch-and-bound* for optimization tasks. Since, the mini-bucket scheme computes bounds (upper or lower) of the exact quantities, these bounds can be used as heuristic functions to guide search algorithms and for pruning the search space. In other words, rather than stopping with the first solution found (which is a lower bound), as it is done in the forward step of *mbe-mpe*, we can continue searching for better solutions, while using the mini-bucket functions to guide and prune the search. This approach was explored extensively in the recent decade and demonstrated very good results both for probabilistic optimization tasks such as MPE as well as for constraint optimization problems [39, 48].

Finally in the context of sum-product queries such as belief computation and probability of evidence we can view the mini-bucket's output as an approximate Bayesian network. Thus, instead of generating larger and larger function by increasing mini-bucket's i-bound, we can use this networks as a basis for sampling schemes, generating what is known as the proposal distribution for importance sampling. For more details on this direction see [62, 78, 33, 32].

8.6.2 Heuristic for partitionings

Clearly, there are many ways to partition a bucket into a collection of mini-buckets having a give parameters i and m. These partitionings can be guided by the graph, or by the content of the actual functions, aiming to minimize the error incurred by a particular partition. It is clear that optimizing the partition strategy is hard and therefore a greedy heuristic is most appropriate.

One popular partitioning heuristic, was scope-based, relying solely on the functions arguments.

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

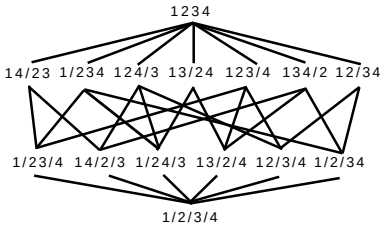


Figure 8.11: Partitioning lattice of bucket $\{f_1, f_2, f_3, f_4\}$. We specify each function by its subindex.

function mini-buckets are decreasingly ordered according to their arity. Then, each mini-bucket is absorbed into the left-most mini-bucket with whom it can be merged. The time and space complexity of `Partition`(B, i) , where B is the partitioned bucket, using the SCP heuristic is $O(|B| \log(|B|) + |B|^2)$ and $O(exp(i))$, respectively. The scope-based heuristic is is quite fast, its shortcoming is that it does not consider the actual information contained in each function.

Recently, 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 d. Therefore, the partition task is to find an i-partition Q^* of B such that $Q^* = \arg\min_Q d(g^Q, g)$. Measure considered includes *log relative error*, *maximum log relative error*, *KL divergence* and *absolute error*. For example:
- *Log relative error*:

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

- *Max log relative error*:

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

We can organize the space of partitions in a lattice using the *refinement* relation since it yields a partial order. Each partition Q of bucket B is a vertex in the lattice. There is an upward edge from Q to Q' if Q' results from merging two mini-buckets of Q in which case Q' is a *child* of Q. The set of all children of Q is denoted by $ch(Q)$. The *bottom*

partition in the lattice is $Q^{\perp}$ while the *top* partition is $Q^{\top}$. For any two partitions Q and Q', if Q' is a descendent of Q then $g^{Q'}$ is clearly tighter than g^{Q} . Namely,

Example 8.6.1 Figure 8.11 shows the partitioning lattice of bucket $\mathcal{B} = \{f_1, f_2, f_3, f_4\}$. Consider a bucket $\mathcal{B}_x = \{f_1, f_2, f_3, f_4\}$. Its Hasse diagram in depicted in Figure 8.11. 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, yields a computationally hard task, scheme focused on depth-first greedy traversals only, are preferred. By looking at the error that occur when considering the options of combining two functions into a single mini-bucket versus keeping them separate, a guiding heuristic function along the lattice can be used. The traversal can be guided by a heuristic function h defined on a partition Q and its child partition Q', denoted $Q \rightarrow Q'$. The local distance heuristics derived from the above distance measures yield *content-based* local partitioning heuristics. For more information see [**?**]

Generalized mini-bucket. The idea of scheduling the mini-buckets that are processed can be relaxed and does not have to be regimented along the original buckets. In other words, we can process 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 C and then go back and process another mini-bucket of B and so on. This alternative schedule is visible once we reason about partitioning heuristics as node duplications. When we have a relaxed network we can process it by any means, not necessarily by bucket-elimination. This observation provide a richer collection of bounding schemes that can be studied.

8.6.3 Current mini-bucket extensions

Current focus is on augmenting the mini-bucket scheme with mini-bucket re-parameterizations ideas, also known as soft-arc consistency. The idea is to shift some of the function content in between mini-buckets while maintaining an equivalent representation of the problem,

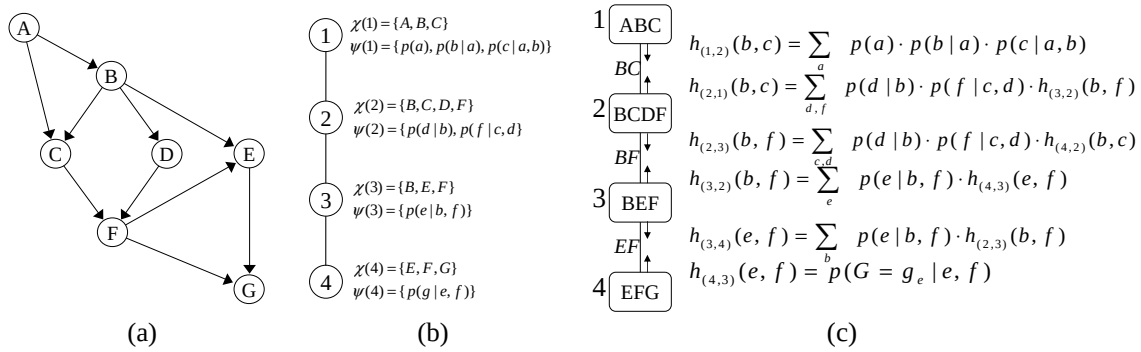


Figure 8.12: (a) A belief network; (b) A join-tree decomposition; (c) Execution of CTE-BU.

in a way that makes the mini-bucket functions more balanced. An orthogonal idea applicable to the sum-product cases is of *weighted mini-buckets* [**?**], which allows a richer collection of choices of mini-bucket as being processed by optimization marginalization or by summation. As described above, currently one mini-bucket should be processed by summation while the rest by optimization (min or max). We can instead associate weights with each mini-bucket in a particular manner. The weighted scheme can be instantiated int the selection of max or min mini-buckets as their special cases, and can guide an optimal selection of mini-buckets to sum over.

8.7 Partition-Based Mini-Clustering

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

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

two conditions of running intersection property and containment.

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. We partition the cluster 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 by CTE-BU $h_{(u,v)} = \sum_{elim(u,v)} \prod_{f \in \psi(u)} f$; we can divide the functions of $\psi(u)$ into p mini-clusters $mc(k)$, and rewrite $h_{(u,v)} = \sum_{elim(u,v)} \prod_{f \in \psi(u)} f = \sum_{elim(u,v)} \prod_{k=1}^{p} \prod_{f \in mc(k)} f$. By migrating the summation operator into each mini-cluster, yielding $\prod_{k=1}^{p} \sum_{elim(u,v)} \prod_{f \in mc(k)} f$, we get an upper bound on $h_{(u,v)}$. The resulting algorithm is called MC-BU(i).

Consequently, the combined functions are approximated via mini-clusters, as follows. Suppose $u \in V$ has received messages from all its neighbors other than v (the message from v is ignored even if received). The functions in $cluster_v(u)$ that are to be combined are partitioned into mini-clusters $\{mc(1), \ldots, mc(p)\}$, each one containing at most i variables. Each mini-cluster is processed by summation over the eliminator, and the resulting combined functions as well as all the individual functions are sent to v.

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

Algorithm MC-BU for upper bounds can be obtained from CTE-BU by replacing step 2 of the main loop and the final part of computing the upper bounds on the joint belief by the procedure given in Figure 8.23.

The partitioning of clusters to mini-clusters can be done in an identical manner to partitioning buckets into mini-buckets as discussed earlier.

Example 8.7.1 Figure 8.13 shows the trace of running MC-BU(3) on the problem in Figure 8.12. 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 $h_{(1,2)}1(b, c) = \sum_a p(a) \cdot p(b|a) \cdot p(c|a, b)$ is computed and the message $H_{(1,2)} = \{h_{(1,2)}1(b, c)\}$ is sent to node 2. Now, node 2 can send its message to node 3. Again, there are no individual functions.

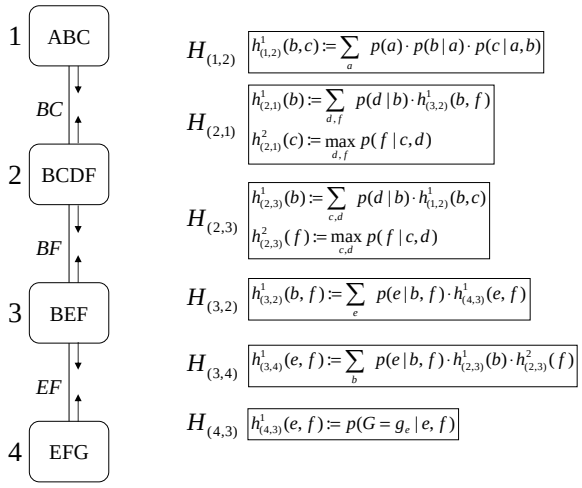


Figure 8.13: Execution of MC-BU for $i = 3$

Cluster 2 contains 4 variables, $\chi(2) = \{B, C, D, F\}$, and a partitioning is necessary: MC-BU(3) can choose $mc(1) = \{p(d|b), h_{(1,2)}(b, c)\}$ and $mc(2) = \{p(f|c, d)\}$. The combined functions $h_{(2,3)}1(b) = \sum_{c,d} p(d|b) \cdot h_{(1,2)}(b, c)$ and $h_{(2,3)}2(f) = max_{c,d}p(f|c, d)$ are computed and the message $H_{(4,3)} = \{h_{(2,3)}1(b), h_{(2,3)}2(f)\}$ is sent to node 3. The algorithm continues until every node has received messages from all its neighbors. An upper bound on $p(a, G = g_e)$ can now be computed by choosing cluster 1, which contains variable A. It doesn't need partitioning, so the algorithm just computes $\sum_{b,c} p(a) \cdot p(b|a) \cdot p(c|a, b) \cdot h_{(2,1)}1(b) \cdot h_{(2,1)}2(c)$. Notice that unlike CTE-BU which processes 4 variables in cluster 2, MC-BU(3) never processes more than 3 variables at a time. □

It is easy to see that,

Theorem 8.7.2 *MC-BU(i) computes an upper bound on the joint probability $P(X, e)$ of each variable and each of its values.*

Theorem 8.7.3 (Complexity of MC-BU(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-BU(i) is $O(n \cdot hw^* \cdot d^i)$, where n is the number of variables, d is the maximum domain size of a variable and $hw^* = max_{u\in T}|\{f \in P|scope(f) \cap \chi(u) \neq \varphi\}|$, which bounds the number of mini-clusters.*

Semantics of Mini-Clustering.

The mini-clustering generalizes the mini-bucket scheme which was shown to have the semantics of relaxation via *node duplication* [38, **?**]. We extend it to mini-clustering by showing how it can apply as is to messages flow in one direction (inward, from leaves to root), as follows. Given a tree-decomposition D, when computing a function $h_{(u,v)}$ (the message that cluster u sends to cluster v), cluster u is partitioned into p mini-clusters $u_1, ..., u_p$, which are first computed independently and then multiplied together. Instead consider a different decomposition D', which is just like D, with the exception that (a) instead of u, it has clusters $u_1, ..., u_p$, all of which are children of v, and each variable appearing in more than a single mini-cluster becomes a new variable, (b) each child w of u (in D) is a child of u_k (in D'), such that $h_{(w,u)}$ (in D) is assigned to u_k (in D') during the partitioning. Note that D' is not a legal tree-decomposition relative to the original variables since it violates the connectedness property: the mini-clusters $u_1, ..., u_p$ contain variables $elim(u, v)$ but the path between the nodes $u_1, ..., u_p$ (this path goes through v) does not. However, it is a legal tree-decomposition relative to the new variables. It is straightforward to see that $H_{(u,v)}$ computed by MC-BU(i) on D is the same as $\{h_{(u_i,v)}|i = 1, ..., p\}$ computed by CTE-BU on D' in the direction from leaves to root.

Example 8.7.4 Figure 8.14(a) shows a trace of the bottom-up phase of MC-BU(3) on the network in Figure 8.13. Figure 8.14(b) shows a trace of the bottom-up phase of CTE-BU algorithm on a problem obtained from the problem in Figure 8.13 by splitting nodes D (into D' and D'') and F (into F' and F''). □

The MC-BU algorithm computes an upper bound $\overline{P}(X_i, e)$ on the joint probability $P(X_i, e)$. However, deriving a bound on the conditional probability $P(X_i|e)$ is not easy when the exact value of $P(e)$ is not available. If we just try to divide (multiply) $\overline{P}(X_i, e)$ by a constant, the result is not necessarily an upper bound on $P(X_i|e)$. It is easy to show that normalization with the *mean* operator is identical to normalization of MC-BU output when applying the summation operator in all the mini-clusters.

MC-BU(i) is an improvement over the Mini-Bucket algorithm MB(i), in that it allows the computation of $\overline{P}(X_i, e)$ for all variables with a single run, whereas MB(i) computes

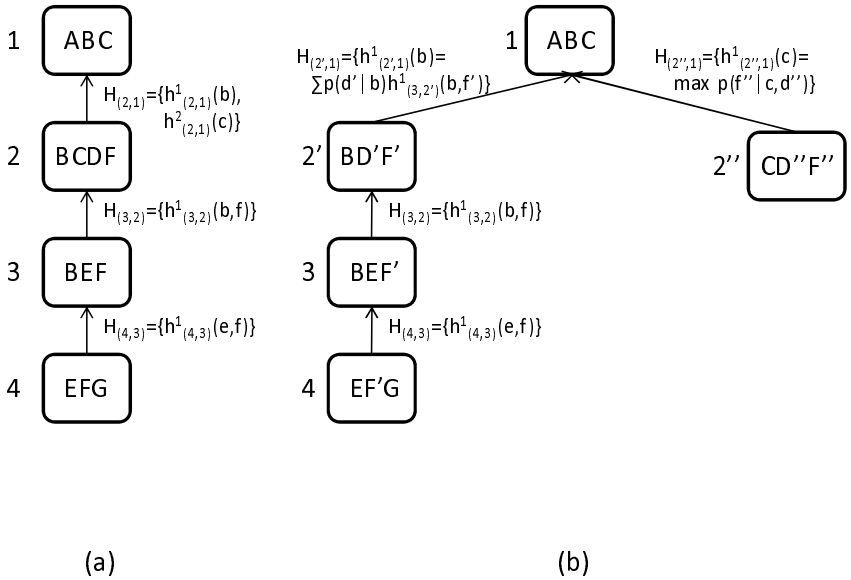


Figure 8.14: Node duplication semantics of MC: (a) trace of MC-BU(3); (b) trace of CTE-BU.

$\overline{P}(X_i, e)$ for just one variable, with a single run [63]. When computing $\overline{P}(X_i, e)$ for each variable, MB(i) has to be run n times, once for each variable (an algorithm we call nMB(i)). In [38] it was demonstrated that MC-BU(i) has up to linear speed-up over nMB(i). For a given i, the accuracy of MC-BU(i) can be shown to be not worse than that of nMB(i).

8.8 Iterative Join-Graph Propagation

Mini-clustering is an anytime algorithm but 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 combine 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* (providing the anytime nature), and the cycles in the graph allow iterating.

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

Definition 8.8.1 (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.*

Definition 8.8.2 (joinwidth) *Let* $D = \langle JG, \chi, \psi \rangle$ *be a join-graph decomposition of a belief network* $\mathcal{B} = \langle X, D, G, P \rangle$. *The* joinwidth *of* D *is* $max_{v \in V} |\chi(v)|$. *The joinwidth of* $\mathcal{B}$ *is the minimum joinwidth over all its join-graph decompositions.*

We will refer to a node and its CPT functions as a *cluster* (note, though, 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 *over-counting*. To reduce this problem we devise a scheme, which avoids cyclicity with respect to any single variable. The algorithm works on arc-labeled join-graphs.

If a graph-decomposition is not arc-minimal it is easy to remove some of its arcs until it becomes arc-minimal. In our preliminary experiments we observed immediately that when applying join-tree propagation on a join-graph iteratively, it is crucial to avoid cycling messages relative to every single variable. The property of arc-minimality is *not* sufficient to ensure such acyclicity though. What is required is that, for every variable X, the arc-subgraph that contains X be a tree.

Example 8.8.3 The example in Figure 8.15a 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. □

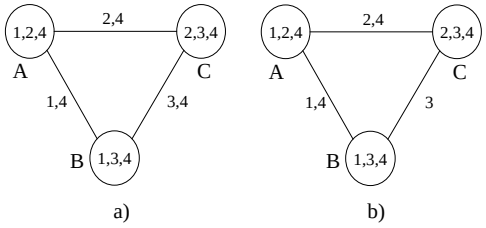


Figure 8.15: An arc-labeled decomposition

To allow more flexible notions of connectedness we refine the definition of join-graph decompositions, when arcs can be labeled with a subset of their separator.

Definition 8.8.4 ((minimal) arc-labeled join-graph decompositions) *An* arc-labeled decomposition *for* $BN =< X, D, G, P >$ *is a four-tuple* $D =< JG, \chi, \psi, \theta >$, *where* $JG = (V, E)$ *is a graph,* χ *and* ψ *associate with each vertex* $v \in V$ *the sets* $\chi(v) \subseteq X$ *and* $\psi(v) \subseteq P$ *and* θ *associates with each edge* $(v, u) \subset E$ *the set* $\theta((v, u)) \subseteq X$ *such that:*

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 8.8.5 (separator, eliminator) *recall that Given two adjacent vertices* u *and* v *of* JG, *the* separator *of* u *and* v *is defined as* $sep(u, v) = \theta((u, v))$, *and the* eliminator *of* u *with respect to* v *is* $elim(u, v) = \chi(u) - \theta((u, v))$.

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

Proposition 8.8.6 *A* minimal edge-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 edge-labeled join-graph is edge-minimal (no edge can be deleted), but not vice-versa. The mini-clustering approximation presented in the previous section works by relaxing the join-tree requirement of exact inference into a collection of join-trees having smaller cluster size. It introduces some independencies in the original problem via node duplication and applies exact inference on the relaxed model requiring only 2 message passings. For the class of IJGP algorithms the idea is to relax the tree-structure requirement and use join-graphs which do not introduce any new independencies, utilizing an iterative message-passing on the resulting cyclic structure.

Indeed, it can be shown that any join-graph of a belief network does not introduce any new independencies to the problem, namely it is an I-map (independency map [58]) of the underlying probability distribution relative to node-separation. Since we plan to use minimally edge-labeled join-graphs to address over-counting problems, the question is what kind of independencies are captured by such graphs.

Definition 8.8.7 (edge-separation in (edge-labeled) join-graphs) *Let $D = \langle JG, \chi, \psi, \theta \rangle$, $JG = (V, E)$ be an edge-labeled decomposition of a Bayesian network $\mathcal{B} = \langle X, D, G, P \rangle$. 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.*

Theorem 8.8.8 *Any edge-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.*

For a proof see [51].

8.8.1 Algorithm IJGP

Applying CTE iteratively to minimal edge-labeled join-graphs yields our algorithm *Iterative Join-Graph Propagation (IJGP)* described in Figure 8.16. One iteration of the algorithm applies message-passing in a topological order over the join-graph, forward and

Algorithm **Iterative Join Graph Propagation (IJGP)**

Input An arc-labeled join-graph decomposition $\langle JG, \chi, \psi, \theta \rangle, JG = (V, E)$ for $\mathcal{B} = \langle X, D, G, P \rangle$. Evidence variables $var(e)$.

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 $h_{(u,v)}$ the message from vertex u to v, $ne_v(u)$ the neighbors of u in JG excluding v.
$cluster(u) = \psi(u) \cup \{h_{(v,u)} | (v, u) \in E\}$.
$cluster_v(u) = cluster(u)$ 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 observed variables:**
   Assign relevant evidence to all $p_i \in \psi(u)$ $\chi(u) := \chi(u) - var(e), \quad \forall u \in V$
2. **Compute individual functions:**
   Include in $H_{(u,v)}$ each function in $cluster_v(u)$ whose scope does not contain variables in $elim(u, v)$. Denote by A the remaining functions.
3. **Compute and send to v the combined function:** $h_{(u,v)} = \sum_{elim(u,v)} \prod_{f \in A} f$.
   Send $h_{(u,v)}$ and the individual functions $H_{(u,v)}$ 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 8.16: Algorithm Iterative Join-Graph Propagation (IJGP)

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 8.8.9 *1. [45] 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. [44] The time complexity of one iteration of IJGP is $O(deg \cdot (n + N) \cdot d^{w^*+1})$ and its space complexity is $O(N \cdot d^{\theta})$, 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, d is the maximum domain size, w^* is the maximum cluster size and θ is the maximum label size.

For proof see properties of CTE in [37].

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. It is precisely these issues that we address in Section **??**. Another explanation will be provided 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, as we discuss later.

Next we will demonstrate that the well-known algorithm IBP is a special case of IJGP.

8.8.2 The Special Case of Iterative Belief Propagation

Iterative belief propagation (IBP) is an iterative application of Pearl's algorithm (that was defined for poly-trees [58]) to any Bayesian network. We will describe IBP as an instance of join-graph propagation over a *dual graph.*

Definition 8.8.10 (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, $E = \{(f_i, f_j) | S_i \cap S_j \neq \varphi\}$. L is a set of labels for the edges, each edge 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 edge-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.*

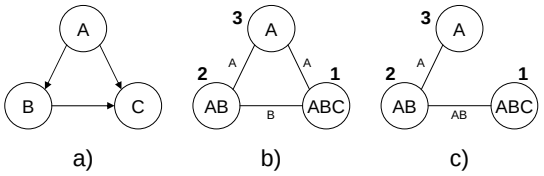


Figure 8.17: a) A belief network; b) A dual join-graph with singleton labels; c) A dual join-graph which is a join-tree

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 8.8.11 *Any Bayesian network has a minimal dual join-graph where each edge 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 p_i is connected to node p_j, $j < i$ if $X_j \in pa(X_i)$. We label the edge between p_j and p_i by variable X_j, namely $l_{ij} = \{X_j\}$. It is easy to see that the resulting edge-labeled subgraph of the dual graph satisfies connectedness. (Take the original acyclic graph G and add to each node its CPT family, namely all the other parents that precede it in the ordering. Since G already satisfies connectedness so is the minimal graph generated.) The resulting labeled graph is a dual graph with singleton labels. ∎

Example 8.8.12 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 8.17a. Figure 8.17b shows a dual graph with singleton labels on the edges. Figure 8.17c 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 IBP algorithm in Figure 8.18. The algorithm is a special case of IJGP. It is easy to see that one iteration of IBP is time and space linear in the

Algorithm **IBP**
Input: An edge-labeled dual join-graph $DG = (V, E, L)$ for a Bayesian network $\mathcal{B} = \langle X, D, G, P \rangle$. 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^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:**

$$h_u^v = \sum_{elim(u,v)} (p_u \cdot \prod_{\{h_i^u, i \in ne_v(u)\}} h_i^u)$$

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,
$P(F_i, e) = \alpha(\prod_{h_i^u, u \in ne(i)} h_i^u) \cdot p_u$;
$P(X_i, e) = \alpha \sum_{scope(u) - \{X_i\}} P(F_i, e)$.

Figure 8.18: Algorithm Iterative Belief Propagation

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) to the exact beliefs.

8.8.3 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.

Given a join-graph decomposition $D = \langle JG, \chi, \psi, \theta \rangle$, the accuracy and complexity on the (iterative) join-graph propagation algorithm depends on the joinwidth of D defined

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 8.19: 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 8.20: Procedure Schematic Mini-Bucket(i).

as $max_{v \in V}|\chi(v)|$. Intuition also suggests that 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 internal width is bounded by i and whose external width is minimized.

We can consider two classes of algorithms. One class is *partition-based*. It starts from a given tree-decomposition and then partitions the clusters until the decomposition has clusters bounded by i. An alternative approach is *grouping-based*. 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 procedure *Schematic Mini-Bucket(i)*, described in Figure 8.20. 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 8.8.13 *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.

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 where variable X is eliminated. All nodes along this chain connect 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

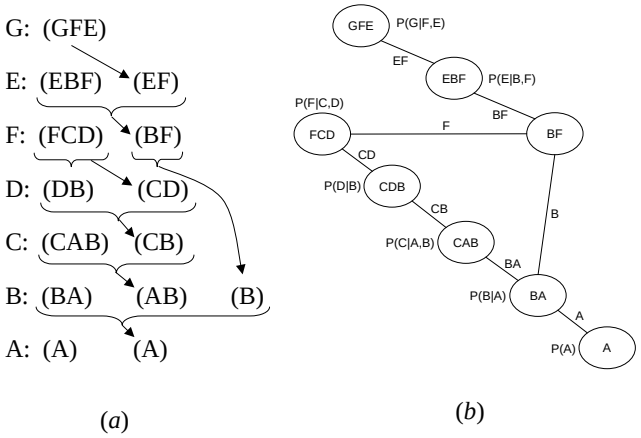


Figure 8.21: Join-graph decompositions.

means variable X must be the variable being eliminated in the bucket of this in-edge. That means 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. ∎

Example 8.8.14 Figure 8.21a shows the trace of procedure schematic mini-bucket(3) applied to the problem described in Figure 8.12a. The decomposition in Figure 8.21b 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 8.8.15 Figure 8.22 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. □

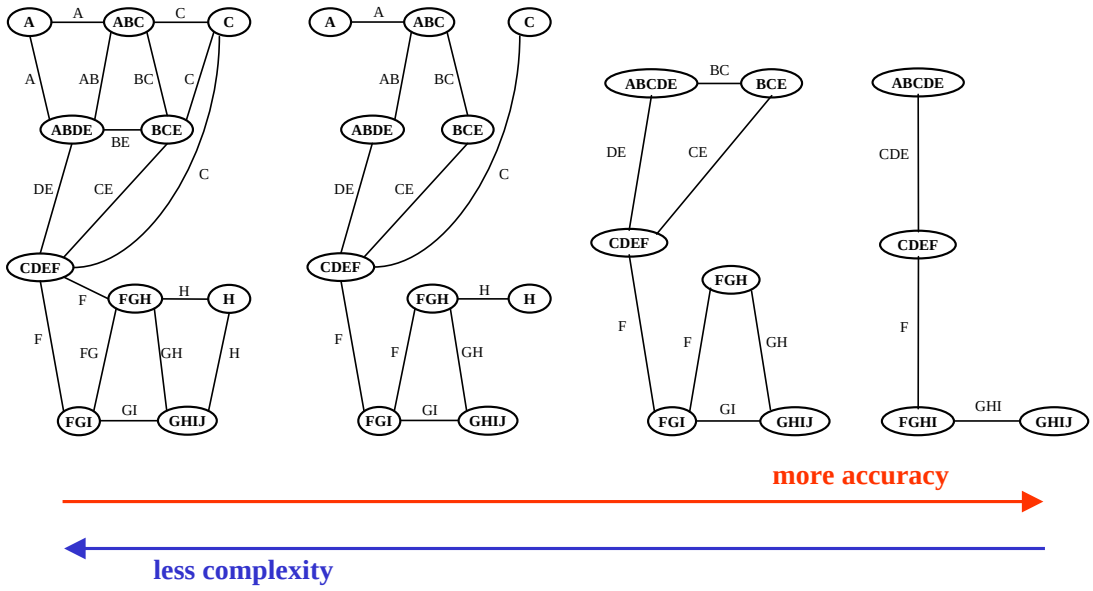


Figure 8.22: Join-graphs.

Procedure **MC for Belief Updating (MC-BU(i))**
Input: A tree decomposition $\langle T, \chi, \psi \rangle T = (V, E)$ for
$\mathcal{B} = \langle X, D, G, P \rangle$. Evidence variables $var(e)$. Accuracy 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 $H_{(u,v)}$ the message sent by vertex u to vertex v, $ne_v(u)$ the neighbors of u in T excluding v.
$cluster(u) = \psi(u) \cup \{H_{(v,u)} | (v, u) \in E\}$.

Compute messages:
For every node u in the cluster tree, T, once u has received messages from all $ne_v(u)$, compute message to node v:

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

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

Figure 8.23: Procedure Mini-Clustering for Belief Updating (MC-BU)

Bibliography

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

[2] D. Allen and A. Darwiche. New advances in inference by recursive conditioning. In *Proceedings of the 19th Conference on uncertainty in Artificial Intelligence (UAI03)*, pages 2–10, 2003.

[3] S. Arnborg and A. Proskourowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete and Applied Mathematics*, 23:11–24, 1989.

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

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

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

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

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

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

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

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

[12] R. McEliece D. C. MacKay and J. Cheng. Turbo decoding as an instance of pearl's "belief propagation" algorithm. 1996.

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

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

[15] 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.

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

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

[18] R. Dechter. Mini-buckets: A general scheme of generating approximations in automated reasoning. In *IJCAI-97: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1297–1302, 1997.

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

[20] R. Dechter and D. Larkin. Hybrid processing of belief and constraints. *Proceeding of Uncertainty in Artificial Intelligence (UAI01)*, pages 112–119, 2001.

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

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

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

[24] R. Dechter and P. van Beek. Local and global relational consistency. In *Principles and Practice of Constraint programming (CP-95)*, pages 240–257, 1995.

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

[26] Rina Dechter. Tractable structures for constraint satisfaction problems. In *Handbook of Constraint Programming, part I, chapter 7*, pages 209–244. Elsevier, 2006.

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

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

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

[30] 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.

[31] J. Gaschnig. Performance measurement and analysis of search algorithms. Technical Report CMU-CS-79-124, Carnegie Mellon University, 1979.

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

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

[34] 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.

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

[36] R. J. Bayardo Jr. and R. C. Schrag. Using csp look-back techniques to solve real world sat instances. In *14th National Conf. on Artificial Intelligence (AAAI97)*, pages 203–208, 1997.

[37] Vibhav Gogate Kalev Kask, Rina Dechter. Counting-based look-ahead schemes for constraint satisfaction. In *Constraint Programming (CP04)*, pages 317–331, 2004.

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

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

[40] 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.

[41] U. Kjæaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Uncertainty in Artificial Intelligence (UAI'93)*, pages 121–149, 1993.

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

[43] F. R. Kschischang and B.H. Frey. Iterative decoding of compound codes by probability propagation in graphical models. *submitted*, 1996.

[44] J Larrosa, K. Kask, and R. Dechter. Up and down mini-bucket: a scheme for approximating combinatorial optimization tasks. *Submitted*, 2001.

[45] 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.

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

[47] R. Marinescu and R. Dechter. AND/OR branch-and-bound for graphical models. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 224–229, 2005.

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

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

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

[51] R. Mateescu, K. Kask, V. Gogate, and R. Dechter. Join-graph propagation algorithms. *Journal of Artificial Intelligence Research (JAIR)*, 37:279–328, 2010.

[52] R.J. McEliece, D.J.C. MacKay, and J.-F.Cheng. Turbo decoding as an instance of Pearl's belief propagation algorithm. *To appear in IEEE J. Selected Areas in Communication*, 1997.

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

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

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

[56] Jurg Ott. *Analysis of Human Genetics.* Cambridge University Press, 1999.

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

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

[59] P. Prosser. Hybrid algorithms for constraint satisfaction problems. *Computational Intelligence*, 9(3):268–299, 1993.

[60] Jr. R. Bayardo and D. P. Miranker. On the space-time trade-off in solving constraint satisfaction problems. In *Fourteenth International Joint Conference on Artificial Intelligence(I95)*, pages 558–562, 1995.

[61] 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.

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

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

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

[65] R.G.Gallager. A simple derivation of the coding theorem and some applications. *IEEE Trans. Information Theory*, IT-11:3–18, 1965.

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

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

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

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

[70] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423,623–656, 1948.

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

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

[73] C.E. Leiserson T. H. Cormen and R.L. Rivest. In *Introduction to algorithms*. The MIT Press, 1990.

[74] 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.

[75] J.A. Tatman and R.D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 365–379, 1990.

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

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

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

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