Chapter 1

Bounding Inference Approximations

This chapter presents a class of approximation algorithms that bound inference by bounding the dimensionality of dependencies created by inference algorithms. This yields a parameterized scheme, called mini-buckets, that offers adjustable trade-off between accuracy and efficiency. The mini-bucket approach to optimization problems, such as finding the most probable explanation (MPE) in Bayesian networks, generates both an approximate solution and a bound on the solution quality.

As noted, automated reasoning tasks over graphical models such as constraint satisfaction and optimization, probabilistic inference, decision-making, and planning are generally hard (NP-hard). This was the focus of Chapter **??** where we introduced algorithms that are tractable if the problem's graph has a small treewidth. When a problem has a high treewidth we must resort to approximations.

Although approximation within given error bounds is also known to be NP-hard [58, 70], there are approximation strategies that work well in practice. One approach advocates *anytime algorithms*. These algorithms can be interrupted at any time producing the best solution found thus far [15, 8].

In this chapter we present a family of parameterized algorithms, called mini-bucket approximations (or elimination) that allow a flexible trade-off between accuracy and efficiency and that can be combined in an anytime algorithm. We provide conditions under which the approximation algorithms find an exact solution and identify regions of good performance.

The class of *mini-bucket* approximation algorithms we propose imports the idea of *local inference* from constraint networks to probabilistic reasoning and combinatorial optimization using the *bucket-elimination* framework. As we showed in Chapter **??** Bucket-elimination is a unifying algorithmic scheme that generalizes non-serial dynamic programming to enable complex problem-solving and reasoning activities. Among the algorithms

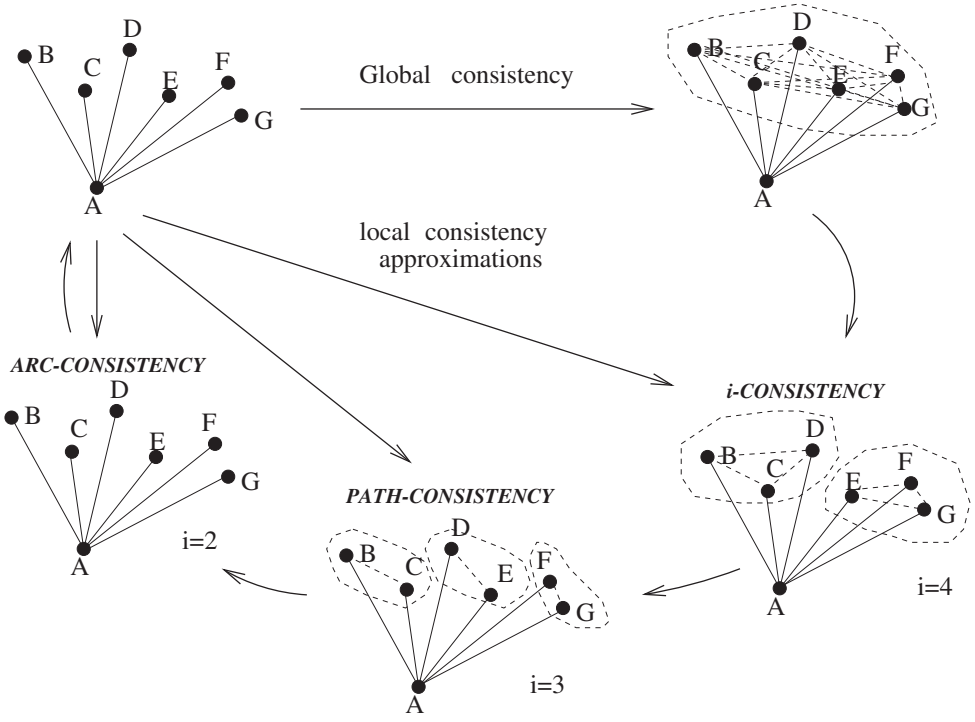


Figure 1.1: From global to local consistency: graph aspects of algorithm i-consistency and two particular cases, path-consistency (i=3) and arc-consistency (i=2).

that can be expressed as bucket-elimination are *directional-resolution* for propositional satisfiability [26], *adaptive-consistency* for constraint satisfaction [24], *Fourier* and *Gaussian elimination* for linear inequalities [48], *dynamic-programming* for combinatorial optimization [6], as well as many algorithms for probabilistic inference [18].

In all these areas problems are represented by a set of variables and by a set of dependencies (e.g., constraints, cost functions, and probabilities) that can be captured by a graph. The algorithms infer and record new dependencies which amounts to adding new edges to the graph. Generally, representing a dependence among k variables (k is called *arity* of a dependence) requires enumerating $O(exp(k))$ tuples. As a result, the complexity of inference is time and space exponential in the arity of the largest dependence recorded which corresponds to the size of largest clique created in the graph and is known as *induced-width*.

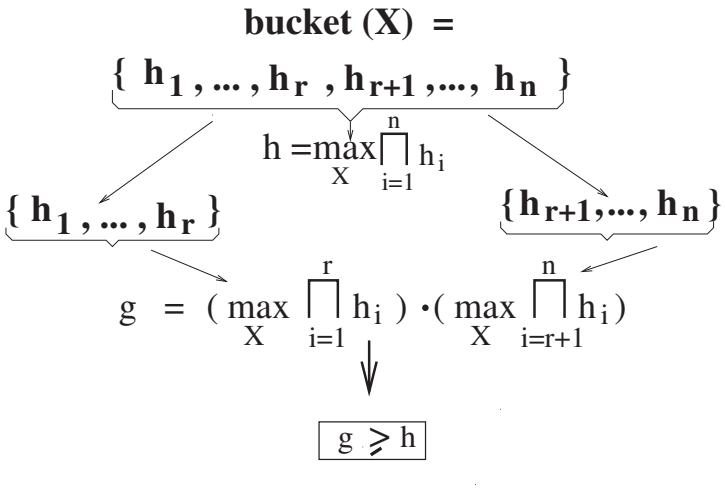


Figure 1.2: The idea of mini-bucket approximation.

1.1 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.

Since the MPE task is NP-hard and since complete algorithms (such as the *cycle cutset* technique, *join-tree-clustering* [60] and bucket elimination [16]) work well only on relatively sparse networks, approximation methods are necessary. Researchers investigated several approaches for finding MPE. The suitability of Stochastic Local Search (SLS) algorithms for MPE was studied in the context of medical diagnosis applications [62] and, more recently, in [41]. Best-First search algorithms were proposed [78] as well as algorithms based on linear programming [71].

In this paper, we propose approximation algorithms based on bucket elimination. Consider the bucket-elimination algorithm *elim-mpe*. Since the complexity of processing a bucket depends on the number of arguments (arity) of the functions being recorded, we propose to approximate 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 *elim-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, we can compute a new function $g^p = \Pi_{i=1}^{t} \max_{X_p} h_i$, that is an upper bound on h^p. Procedurally it means that maximization is applied separately to each function, requiring less computation.

The idea of mini-bucket partitioning is demonstrated in Figure 1.2, 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 *elim-mpe* computes $h^p = \max_{X_p} \Pi_{i=1}^t h_i$, which can be rewritten as $h^p = \max_{X_p} \Pi_{l=1}^r \Pi_{l_i} h_{l_i}$. By migrating maximization into each mini-bucket we can compute: $g_Q^p = \Pi_{l=1}^r \max_{X_p} \Pi_{l_i} h_{l_i}$. The new functions $\max_{X_p} \Pi_{l_i} h_{l_i}$ are 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 maximized product generated in the first bucket is an upper bound on the MPE probability. A lower bound can also be computed as the probability of a (suboptimal) assignment found in the forward step of the algorithm. Clearly, as the mini-buckets get smaller, both complexity and accuracy decrease.

Definition 1.1.1 *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 1.1.2 *If Q'' is a refinement of Q' in bucket$_p$, then $h^p \leq g_{Q'}^p \leq g_{Q''}^p$.*

The mini-bucket elimination (*mbe*) algorithm for finding MPE, *mbe-mpe(i,m)*, is described in Figure 1.3. It can use two input parameters that control the mini-bucket partitioning.

Definition 1.1.3 ((i,m)-partitioning) *Let H be a collection of functions $h_1, ..., h_t$ defined on scopes $S_1, ..., S_t$, respectively. We say that a function f is* subsumed *by a function h if any argument of f is also an argument of h. A partitioning of $h_1, ..., h_t$ is* canonical *if any function f subsumed by another function is placed into the bucket of one of those subsuming functions. A partitioning Q into mini-buckets is an (i, m)-partitioning if and only if (1) it is canonical, (2) at most m non-subsumed functions are included in each mini-bucket, (3) the total number of variables in a mini-bucket does not exceed i, and (4) the partitioning is* refinement-maximal*, namely, there is no other (i, m)-partitioning that it refines.*

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,

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 $L = P(\bar{x}^a)$.

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

Proposition 1.1.4 *If the bound i on the number of variables in 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.*

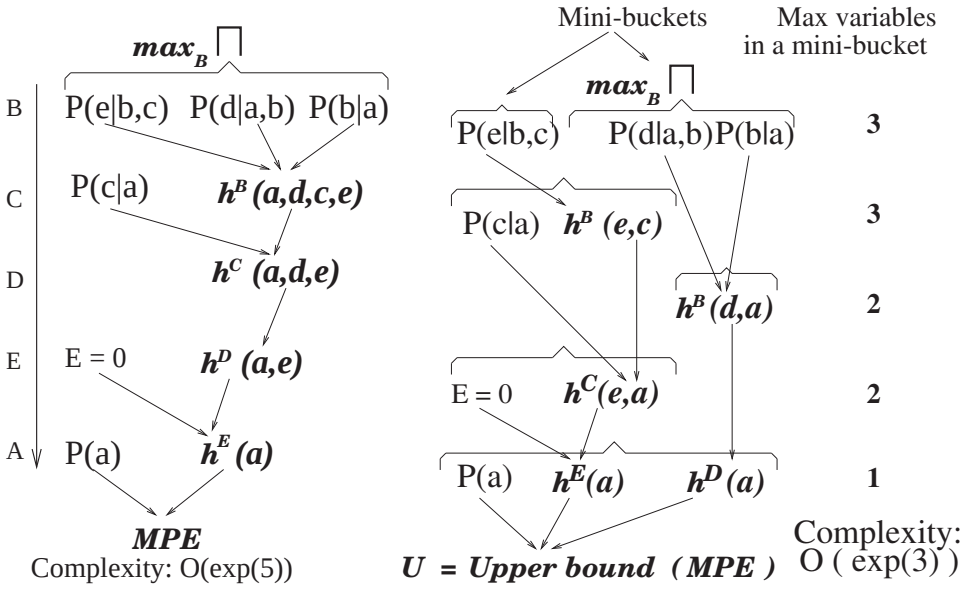
Exercise: Prove proposition 1.1.2.

Although the two parameters i and m are not independent they do allow a variety of partitioning schemes reacher than using i or m alone. The properties of the mini-bucket algorithms are summarized in the following theorem.

Theorem 1.1.5 (mbe-mpe complexity) *Algorithm* mbe-mpe*(i, m) computes an upper bound on the MPE. its time and space complexity is $O(n \cdot exp(i))$ where $i \leq n$.*

We will prove the theorem later (section 1.6) 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.



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

Figure 1.4: Comparison between (a) *elim-mpe* and (b) *mbe-mpe(3,2)*.

Example 1.1.6 Figure 1.4 compares algorithms *elim-mpe* and *mbe-mpe(i,m)* where $i = 3$ and $m = 2$ over the network in Figure **??**a along the ordering $o = (A, E, D, C, B)$. The exact *elim-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, is 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 1.4b (the (3,2)-partitioning is 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.

By design, *mbe-mpe(3,2)* does not produce functions on more than 2 variables, while the exact algorithm *elim-mpe* records a function on 4 variables. □

In summary, algorithm *mbe-mpe(i,m)* computes an interval $[L, U]$ containing the MPE value where U is the upper bound computed by the backward phase and L is the probability of the returned assignment.

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

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

Clearly the bounds U and L for MPE are very close to zero when the evidence $\bar{e}$ is unlikely, however the ratio between the upper and the lower bound is not dependent on $P(\bar{e})$. As we will see next, approximating conditional probabilities using bounds on joint probabilities is more problematic for belief updating.

1.1.1 The mini-bucket semantics

The Mini-Bucket computation can be viewed as relaxation in the following sense. For each bucket and its partitioning into mini-buckets, a variable in the original problem is replaced by a set of new variables in the relaxed problem, each corresponding to a single mini-bucket, and each function in the original problem is associated with the copy of the variable in the relaxed problem 40 corresponding to its mini-bucket in the original problem. For example, the Mini-Bucket trace in Figure 5b, corresponds to solving exactly by full bucket-elimination the following relaxation of the problem in Figure 2. 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 relaxed problem. The relaxed problem has a smaller width and can be solved more efficiently, yielding a bound (upper or lower) as expected.

1.2 Mini-bucket approximation for belief updating

As shown in Chapter **??**, the bucket elimination algorithm *elim-bel* for belief assessment is similar to *elim-mpe* except that maximization is replaced by summation and no value assignment is generated. Algorithm *elim-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 **??**).

The mini-bucket idea used for approximating MPE can be applied to belief updating in a similar way. Let $Q\prime = \{Q_1, ..., Q_r\}$ be a partitioning of the functions $h_1, ...h_t$

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 $P(x_1, \bar{e})$.
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** the product of functions in the bucket of X_1, which is an upper bound on $P(x_1, \bar{e})$ (denoted $g(x_1)$).

Figure 1.5: Algorithm *mbe-bel-max(i,m)*.

(defined over scopes $S_1, ..., S_t$, respectively) in X_p's bucket. Algorithm *elim-bel* computes $h^p : U_p \rightarrow \Re$, where $h^p = \sum_{X_p} \Pi_{i=1}^t h_i$, and $U_p = \cup_i S_i - \{X_p\}$. Note that $h^p = \sum_{X_p} \Pi_{i=1}^t h_i$, can be rewritten as $h^p = \sum_{X_p} \Pi_{l=1}^r \Pi_{l_i} h_{l_i}$. If we follow the MPE approximation precisely and migrate the summation operator into each mini-bucket, we will compute $f_{Q\prime}^p = \Pi_{l=1}^r \sum_{X_p} \Pi_{l_i} h_{l_i}$. This, however, is an unnecessarily large upper bound of h^p in which each $\Pi_{l_i} h_{l_i}$ is bounded by $\sum_{X_p} \Pi_{l_i} h_{l_i}$. 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, which yields $g_{Q\prime}^p = (\sum_{X_p} \Pi_{1_i} h_{1_i}) \cdot (\Pi_{l=2}^r \max_{X_p} \Pi_{l_i} h_{l_i})$. In summary, an upper bound g^p of h^p can be obtained by processing one of X_p's mini-buckets by summation and the rest by maximization.

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

A lower bound on the belief, or its mean value, can be obtained in a similar way. Algorithm *mbe-bel-max(i,m)* that uses the *max* elimination operator is described in Figure 1.5. Algorithms *mbe-bel-min* and *mbe-bel-mean* can be obtained by replacing the operator *max* by *min* and by *mean*, respectively.

1.2.1 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 $P(x_1|\bar{e})$. In principle, we can derive a lower bound, f, on $P(\bar{e})$ using *mbe-bel-min* (in this case, the observed variables initiate the ordering), and then compute $\frac{g(x_1)}{f}$ as an upper bound on $P(x_1|\bar{e})$. This however is likely to make the bound quite weak due to compounded error. In many practical scenarios, however, we are interested in the ratio between the belief in two competing values of X_1. Since $P(x_i, e)/P(x_j, e) = P(x_i|e)/P(x_j|e)$, the ratio between the upper bounds of the respective join probabilities can serve as a good comparative measure between the conditional probabilities as well.

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 $\frac{L_i}{P(\bar{e})} \leq P(X_1|\bar{e}) \leq 1$,

so that the upper bound is trivial. Finally, note there is no bound for $g_{mean}(x_i)$, and therefore, the approximation of $\frac{g_{mean}(x_i)}{\sum_{x_1} g_{mean}(x_1)}$ can also be a lower or an upper bound of the exact belief. 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.

1.3 Mini-bucket elimination for MAP

Algorithm *elim-map* for computing the MAP is a combination of *elim-mpe* and *elim-bel* as we have shown in Chapter **??**; some of the variables are eliminated by summation, while the others by maximization. The MAP task is generally more difficult than MPE and belief updating [**?**]. From variable elimination perspective it restricts the possible variable orderings and therefore may require higher w^* which implies higher complexity.

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

$$\bar{a}_k^{map} = \arg\max_{\bar{a}_k} P(\bar{a}_k|\bar{e}) = \arg\max_{\bar{a}_k} \frac{\sum_{\bar{x}_{k+1}^n} \Pi_{i=1}^n P(x_i, \bar{e}|x_{pa_i}^-)}{P(\bar{e})} \tag{1.1}$$

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 $P(\bar{a}_k, \bar{e})$. Namely, as before, we have $P(\bar{a}_k|\bar{e}) = \frac{P(\bar{a}_k,\bar{e})}{P(\bar{e})}$. Thus, we define $MAP = P(\bar{a}_k, \bar{e})$ and 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, *elim-map* [16, 21], assumes only orderings in which the hypothesized variables appear first and thus are processed last by the algorithm (this restricted ordering implies increased complexity as remarked above) . The algorithm has a backward phase as usual but its forward phase is relative to the hypothesis variables only. The application of the mini-bucket scheme to *elim-map* for deriving an upper bound 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 is eliminated by summation, we apply the rule we have in *mbe-bel-max* in which one 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 1.6.

Deriving a lower bound on the MAP is no longer a simple extension of *mbe-map* as we observed for MPE. Once *mbe-map* terminates, we have an upper bound and we

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

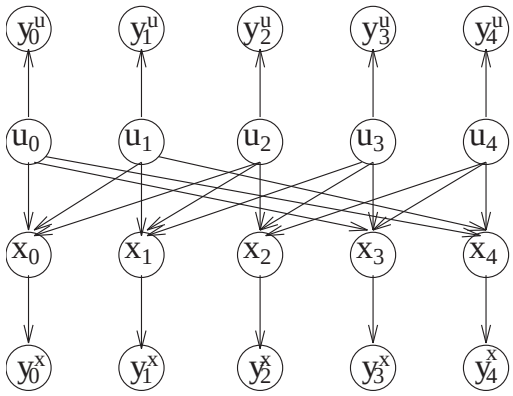


Figure 1.7: Belief network for a linear block code.

can compute an assignment to the hypothesis variables. While the probability of this assignment is a lower bound for the MAP, obtaining this probability is no longer possible by a simple forward step over the generated buckets. It requires an exact inference, or a lower bound approximation. 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 that take into account the original functions and only those computed by *mbe-bel-min*.

Example 1.3.1 We will next demonstrate the mini-bucket approximation for MAP on an example inspired by *probabilistic decoding* [13, 46] [1]. Consider a belief network which describes the decoding of a *linear block code*, shown in Figure 1.7. 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):
$bucket(X_0) = P(y_0^x|X_0), P(X_0|U_0, U_1, U_2),$
$bucket(X_1) = P(y_1^x|X_1), P(X_1|U_1, U_2, U_3),$
$bucket(X_2) = P(y_2^x|X_2), P(X_2|U_2, U_3, U_4),$

[1] Probabilistic decoding is discussed in more details in Section 1.8.4.

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

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 $P(U_1, \bar{e})/P(U_2, \bar{e})$. □

1.4 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 [6]. Cost networks is a general model encompassing constraint-satisfaction, and constraint-optimization in general. 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 an explicit treatment see [19]. 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 **??**. 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.

1.5 A unified presentation of mbe

[to complete]

1.6 Complexity and tractability

1.6.1 The case of low induced width

All mini-bucket algorithms have similar worst-case complexity bounds and completeness conditions. 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 [19][2]. Theorem 1.1.5 applies to all mini-bucket algorithms:

Theorem 1.6.1 *Algorithm* mbe(i,m) *takes* $O(r \cdot exp(i))$ *time and space, where* r *is the*

[2]Note that in case of optimization task, *mbe(i,m)* takes as an input a set of cost functions rather than a set of probability functions [19].

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

number of input functions[3]*, and where* $|F|$ *is the maximum scope of any input function,* $|F| \leq i \leq n$*. For* $m = 1$*, the algorithm is time and space* $O(r \cdot exp(|F|))$*.*

Proof: We can associate a bucket-elimination or a mini-bucket elimination algorithm with a *computation tree* where leaf nodes correspond to the original input functions (CPTs or cost functions), and each internal node v corresponds to the result of applying an elimination operator (e.g., product followed by summation) to the set of node's children, $ch(v)$ (children correspond to all functions in the corresponding bucket or mini-bucket). We can compress the computation tree so that each node having a single child will be merged into one node with its parent, so that the branching degree in the resulting tree is not less than 2. Computing an internal node that is a compressed sequence of single-child nodes takes O(exp(—F—) time and space since it only requires a sequence of elimination operations over a single function which can be accomplished in one pass through the tuples (accumulating appropriate running summations over the relevant variables). The

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

cost of computing any other internal node v is $O(|ch(v)| \cdot exp(i))$ where $|ch(v)| \leq m$ and i bounds the resulting scope size of generated functions. Since the number of leaf nodes is bounded by r, the number of internal nodes in the computation tree is bounded by r as well (since the branching factor of each internal node is at least 2). Thus the total amount of computation over all internal nodes in the computation tree is time and space $O(r \cdot exp(i))$ in general, which becomes to $O(n \cdot exp(i))$ for belief networks. ∎

The above proof, suggested in [**?**], refines the original proof given in [27].

We next identify cases for which the mini-bucket scheme coincides with the exact algorithm, and is therefore complete.

Theorem 1.6.2 *Given an ordering of the variables, o, algorithm* mbe(i, n) *applied along o is complete for networks having* $w_o^* \leq i$.

Proof: The claim trivially follows from the observation that each full bucket satisfies the condition of being an (i, n)-partitioning and it is the only one which is refinement-maximal. ∎

1.6.2 The case of mini-bucket(n,1)

Another case is *mbe(n,1)* which allows only one non-subsumed function in a mini-bucket. It is easy to see that *mini-bucket(n,1)* is complete for polytrees if applied along some *legal* orderings. A *legal ordering* of a polytree (see Figure 1.9) is one where (1) all evidence nodes appear last in the ordering, and (2) among the remaining variables, each child node appears before its parents and all the parents of the same family are consecutive in the ordering. Such an ordering is always feasible for polytrees, and using this ordering, each bucket contains only one non-subsumed function. Clearly, algorithm *mini-bucket(n,1)* is complete along such orderings and is therefore complete for polytrees.

In summary,

Theorem 1.6.3 *Given a polytree, there exists an ordering o such that algorithm* mbe(n,1) *finds an exact solution in time and space* $O(n \cdot exp(|F|))$, *where* $|F|$ *is the largest scope size of any input function.*

Example 1.6.4 Consider a legal ordering $o = (X_1, U_3, U_2, U_1, Y_1, Z_1, Z_2, Z_3)$ of the polytree in Figure 1.9a, where the last four variables Y_1, Z_1, Z_2, Z_3 in the ordering are observed. Processing the buckets from last to first, after the last four buckets were already processed as observation buckets, we get (observed values shown in low-case):
$bucket(U_1) = P(U_1), P(X_1|U_1, U_2, U_3), P(z_1|U_1),$

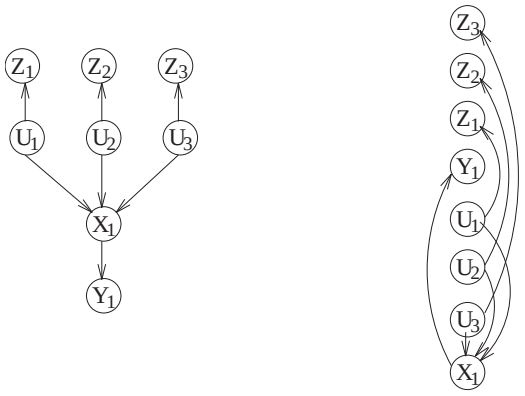


Figure 1.9: (a) A polytree and (b) a legal ordering, assuming that nodes Z_1, Z_2, Z_3 and Y_1 are observed.

$bucket(U_2) = P(U_2), P(z_2|U_2)$,
$bucket(U_3) = P(U_3), P(z_3|U_3)$
$bucket(X_1) = P(y_1|X_1)$.
It is easy to see that the only legal partitionings correspond to full buckets.

□

Note also that on polytrees, *mbe(n,1)* is similar to Pearl's well-known propagation algorithm. One difference, however, is that Pearl's algorithm records only functions defined on a single variable, while mini-bucket(n,1) may record functions whose scope is at most the size of a family.

1.7 Anytime inference

An important property of the mini-bucket scheme is that it provides an adjustable tradeoff between accuracy of solution and computational complexity. Both the accuracy and the complexity increase with increasing 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 [**?**, **?**, 15, 8]. As more time is available, better solutions are guaranteed by such algorithms. In the context of Bayesian networks, anytime algorithms were first considered by the name of *flexible computation* under computational resource constraints [**?**, **?**, **?**].

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 1.10: Algorithm *anytime-mpe(ϵ)*.

One of the first probabilistic anytime inference algorithms was *bounded conditioning* algorithm [**?**] that works by conditioning on a small, high probability cutset instances, including more of the instances as more computational resources become available.

In general, any inference algorithm that adapts to limited computational resources by ignoring some information about the problem, and is able to recover that information incrementally as more resources become available, can be called an anytime inference algorithm [**?**, **?**]. Many approximation schemes can be used by anytime methods since they are based on exploiting only partial information about the problem, e.g. ignoring a subset of "weak" edges [45, **?**], using partial variable assignments [**?**, **?**] (including partial cutset assignments [**?**]), using only a subset of nodes [30], or a subset of (relatively high) probability entries in CPTs [31]. In particular, our mini-bucket scheme exploits partial (bounded) dependencies among the variables. Clearly, an iterative application of such schemes with less restrictions on the amount of information they use, results in anytime inference algorithms that eventually become exact, if sufficient computational resources are available.

Our idea of extending the mini-bucket scheme to an anytime algorithm is to run a sequence of mini-bucket algorithms with increasing values of i and m until either a desired level of accuracy is obtained, or until the computational resources are exhausted. The anytime algorithm *anytime-mpe(ϵ)* for MPE is presented in Figure 1.10. 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 $\epsilon = 0$.

Another anytime extension of the mini-bucket, is to embed it within a complete anytime heuristic search algorithm such as *branch-and-bound*. Since, the mini-bucket approximations computes bounds (upper or lower) of the exact quantities, these bounds can be used as heuristic functions to guide search algorithms as well as for pruning the search space. In other words, rather than stopping with the first solution found, 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 recently and demonstrated great promise both for probabilistic optimization tasks such as MPE as well as for constraint satisfaction problems [42].

1.8 Empirical evaluation

In this section we present some empirical evidence obtained when evaluating the potential of the mbe-mpe scheme. For more details see [**?**].

1.8.1 Methodology

Mbe-mpe(m) denotes the algorithm with an unrestricted i and a varying m, while *mbe-mpe(i)* assumes an unrestricted m and a varying i. Both algorithms use the following brute-force strategy for selecting a mini-bucket partitioning. First, a canonical partitioning is found, i.e. all subsumed functions are placed into mini-buckets of one of their subsuming functions. Then, for *mbe-mpe(m)*, each group of m successive mini-buckets is combined into one mini-bucket. For *mbe-mpe(i)*, we merge the successive canonical mini-buckets into a new one until the total number of variables in that mini-bucket exceeds i. Then the process is repeated for the next group of canonical mini-buckets, and so on. Also, in our implementation, we use a slightly different interpretation of the parameter i. We allow $i < |F|$, where $|F|$ is maximum family size, and bound the number of variables in a mini-bucket by $\max\{i, |F|\}$ rather than by i.

The accuracy of an approximation is measured by the error ratios MPE/L and U/MPE, where U and L are, respectively, the upper and the lower bound on MPE found by *mbe-mpe*, where MPE is the probability of the exact MPE solution found by *elim-mpe*. When computing the exact MPE assignment is infeasible, we report only the

ratio U/L (note that U/L is an upper bound on the error ratios MPE/L and U/MPE). The efficiency gain is represented by the *time ratio* $TR = T_e/T_a$, where T_e is the running time for *elim-mpe* and T_a is the running time for *mbe-mpe*. Remember that for $i > w^*$, *mbe-mpe(i)* coincides with *elim-mpe*.

Random problem generators

The uniform random problem generator takes as an input the number of nodes, n, the number of edges, e, and the number of values per variable, v. An acyclic directed graph is generated by randomly picking e edges and subsequently removing possible directed cycles, parallel edges, and self-loops. Then, for each node x_i and its parents x_{pa_i}, the conditional probability tables (CPTs) $P(x_i|x_{pa_i})$ are generated from the uniform distribution over $[0, 1]$. Namely, each $P(x_i|x_{pa_i})$ is replaced by $P(x_i|x_{pa_i})/\sum_{x_i} P(x_i|x_{pa_i})$.

1.8.2 Random noisy-OR problems

Consider a set of experiments on randomly generated *noisy-OR* networks[4]. A noisy-OR conditional probability table (CPT) is defined on binary-valued nodes as follows: given a child node x, and its parents y_1,...,y_n, each y_i is associated with a *noise parameter* $q_i = P(x = 0|y_i = 1, y_k = 0)$, $k \neq i$. The conditional probabilities are defined as follows [60]:

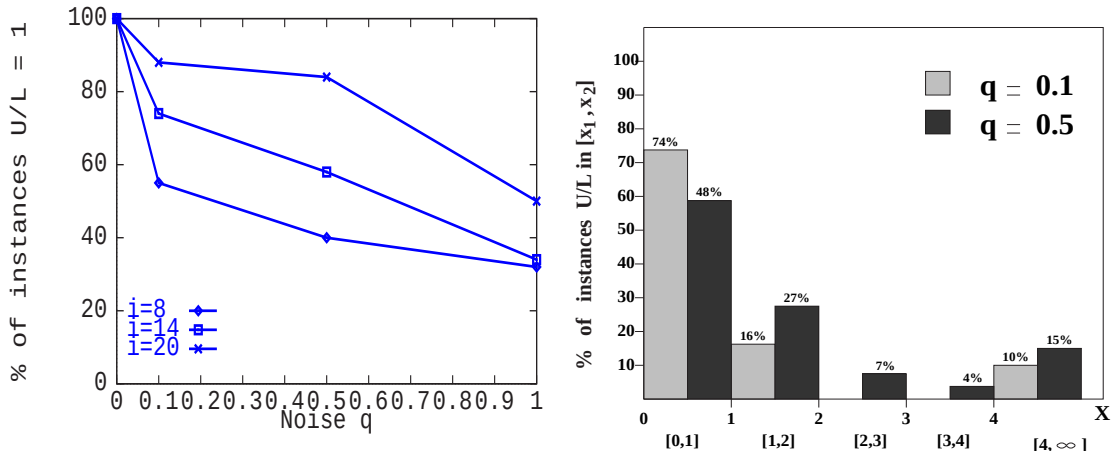
$$P(x|y_1, \ldots, y_n) = \begin{cases} \prod_{y_i=1} q_i & \text{if x = 0,} \\ 1 - \prod_{y_i=1} q_i & \text{if x = 1.} \end{cases} \tag{1.2}$$

Obviously, when all $q_i = 0$, we have a logical OR-gate. The parameter $1 - q_i = P(x = 1|y_i = 1, y_k = 0)$ for $k \neq i$ is also called *link probability*.

We generated random noisy-OR networks using the random graph generator described earlier, and randomly selecting the conditional probabilities for each CPT from the interval $[0, q]$, where q was the bound on the noise parameter.

We present results for *mbe-mpe(i)* on larger networks (Figure 1.11). Algorithm *elim-mpe* was intractable on these problems. The most apparent phenomenon here is that the approximation improves with decreasing noise q, i.e., $U/L \to 1$ for $q \to 0$. In Figure 1.11a, the percentage of instances for which $U/L = 1$ is plotted against q for *mbe-mpe(8)*, *mbe-mpe(14)*, and *mbe-mpe(20)*. When $q = 0$ (deterministic dependence $x \Leftrightarrow y_1 \vee ... \vee y_k$ between a child x and its parents $y_i, i = 1, ..., k$), we observe almost 100% accuracy, which then decreases with increasing q for all values of $i = 8, 14, 20$. One possible explanation is that in the absence of noise we get loosely connected constraint-satisfaction problems

[4]Noisy-OR is an example of *causal independence* [35] which implies that several causes (parent nodes) contribute independently to a common effect (child node).



(a) (b)

Figure 1.11: Results on 200 random noisy-OR networks, each having 50 nodes, 150 edges, and 10 evidence nodes: (a) frequency of problems solved exactly (U/L=1) versus noise q for different values of i; (b) a histogram of U/L for $q = 0.1$ and $q = 0.5$.

which can be easily solved by local constraint propagation techniques coinciding in this case with the mini-bucket scheme.

Notice that the behavior of *mbe-mpe(i)* is "extreme": it is either very accurate, or very inaccurate (Figure 1.11b). This phenomenon is more noticeable for small q.

1.8.3 CPCS networks

We next show results on a set of CPCS networks used in medical diagnosis. CPCS networks are derived from the Computer-based Patient Case Simulation system [57, 64]. CPCS network representation is based on INTERNIST-1 [54] and Quick Medical Reference (QMR) [53] expert systems. The nodes of CPCS networks correspond to diseases and findings. In the original knowledge base, the probabilistic dependencies between the nodes are represented by *frequency weights* that specify the increase in the probability of a finding (child node) given a certain disease (parent node). This representation was later converted into a belief network using several simplifying assumptions: (1) conditional independence of findings given diseases, (2) noisy-OR dependencies between diseases and findings, and (3) marginal independence of diseases [80].

In CPCS networks, the noisy-OR CPTs may also include *leak probabilities*. Namely, given a child node x and its parents y_1,...,y_n, the *leak probability* is defined as $leak = P(x = 1|y_1 = 0, ..., y_n = 0)$. The definition of a noisy-OR CPT is then modified as

follows:

$$P(x = 0|y_1, \ldots, y_n) = (1 - leak) \prod_{y_i=1}^{n} q_i, \tag{1.3}$$

where q_i are noise parameters defined earlier. Some CPCS networks include multivalued variables and *noisy-MAX* CPTs, which generalize noisy-OR by allowing k values per node, as follows:

$$l_i = P(x = i|y_1 = 0, \ldots, y_n = 0), i = 1, \ldots k-1, \text{and}$$

$$P(x = i|y_1, \ldots, y_n) = l_i \prod_{j=1}^{n} q_j^{y_j}, i = 0, \ldots k-2,$$

$$P(x = k-1|y_1, \ldots, y_n) = 1 - \sum_{i=0}^{i=k-2} l_i \prod_{j=1}^{n} q_j^{y_j}, \tag{1.4}$$

where $q_j^{y_j}$ is a noise coefficient for parent j and the parent's value y_j. This definition coincides with the one given by the equation 1.3 for $k = 2$, assuming $l_0 = 1 - leak$, $q_j^0 = 1$, and $q_j^1 = q_j$.

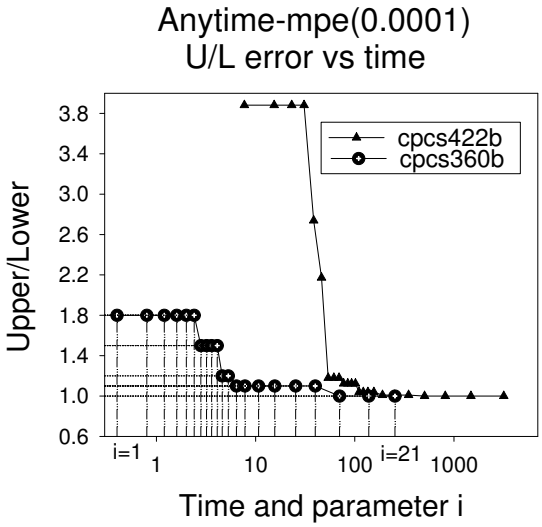
We experimented with both binary (noisy-OR) and non-binary (noisy-MAX) CPCS networks. The noisy-MAX network *cpcs179* (179 nodes, 305 edges) has 2 to 4 values per node, while the noisy-OR networks *cpcs360b* (360 nodes, 729 edges) and *cpcs422b* (422 nodes, 867 edges) have binary nodes (the letter 'b' in the network's name stands for "binary"). Since our implementation used standard conditional probability tables the non-binary versions of the larger CPCS networks with 360 and 422 nodes did not fit into memory. Each CPCS network was tested for different sets of evidence nodes.

Experiments without evidence

We present the results obtained on *cpcs179*, *cpcs360b*, and *cpcs422b* networks assuming no evidence nodes (i.e. there is only one network instance in each case) and using a min-degree elimination ordering.

The results are reorganized in Figure 1.12 from the perspective of algorithm *anytime-mpe(ϵ)*. *Anytime-mpe(ϵ)* runs *mbe-mpe(i)* for increasing i until $U/L < 1 + \epsilon$. We started with $i = 1$ and were incrementing it by 1. We present the results for $\epsilon = 0.0001$ in Figure 1.12. The table compares the time of *anytime-mpe(0.0001)* and of *anytime-mpe(0.1)* against the time of the exact algorithm. We see that the anytime approximation can be an order of magnitude faster.

It is known that approximating posterior marginals is harder when the evidence is unlikely. So, we show performance with *likely evidence* and *random evidence*. A random evidence set of size k is generated by randomly selecting k nodes and assigning value 1

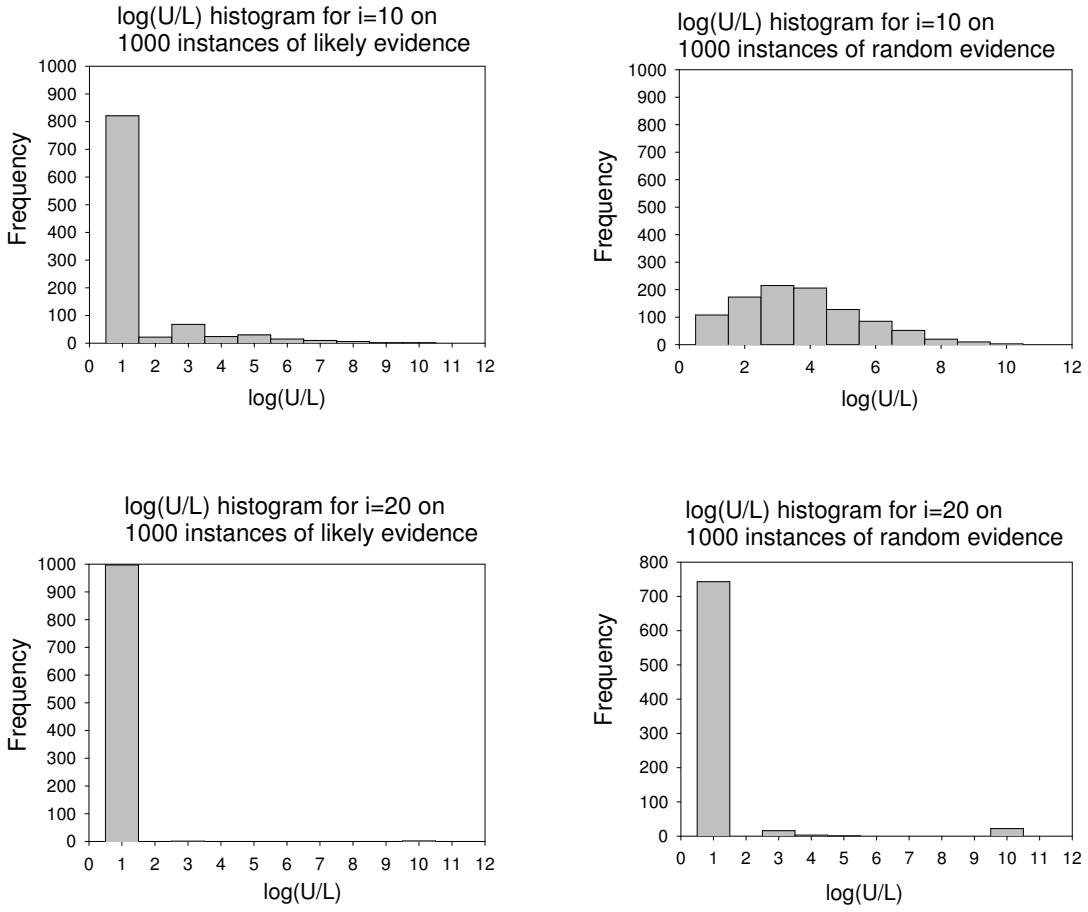


Algorithm	Time (sec)	
	cpcs360	*cpcs422*
elim-mpe	115.8	1697.6
anytime-mpe(ϵ), $\epsilon = 0.0001$	70.3	505.2
anytime-mpe(ϵ), $\epsilon = 0.1$	70.3	110.5

Figure 1.12: *anytime-mpe*(0.0001) on *cpcs360b* and **cpcs422b** networks for the case of no evidence.

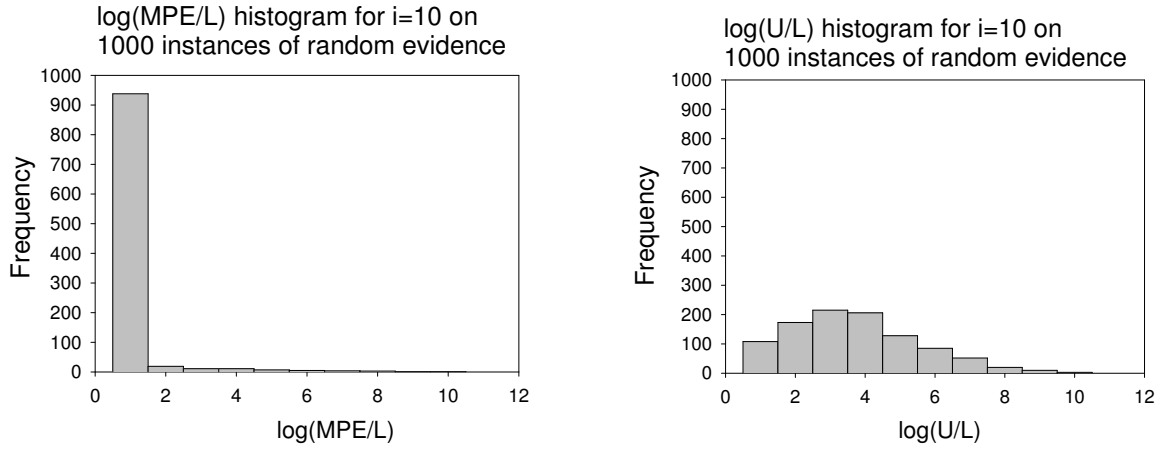
to all of them. This approach usually produces a highly unlikely evidence set that results in low MPE probability. Alternatively, likely evidence can be generated via *ancestral simulation* (*forward sampling* as follows. Starting with the root nodes and following an ancestral ordering where parents precede children, we simulate a value of each node in accordance with its conditional probability table. A given number of evidence nodes is then selected randomly. Ancestral simulation results in relatively high-probability tuples, which produce higher values of MPE than those for random evidence. As we demonstrate below, this has a dramatic impact on the quality of the approximation.

Since the variance of U/L is high, we present histograms of $log(U/L)$ in Figures 1.13 and 1.14, which summarize and highlight our main observations. The accuracy increases for larger values of i (histograms in Figure 1.13 shift to the left with increasing i). As before, we see a dramatic increase in accuracy in case of likely evidence (Figure 1.13), and observe that the lower bound is often much closer to M than the upper bound: MPE/L



(a) Likely evidence (b) Random evidence

Figure 1.13: Histograms of U/L for $i = 10, 20$ on the *cpcs360b* network with 1000 sets of likely and random evidence, each of size 10.



(a) MPE/L (b) U/L

Figure 1.14: Histograms of U/L and MPE/L for $i = 10$ and RANDOM evidence on the *cpcs360b* network. Each histogram is obtained on 1000 randomly generated evidence sets, each of size 10.

is closer to 1 than U/L (see Figure 1.14).

1.8.4 Probabilistic decoding

The purpose of *channel coding* is to provide reliable communication through a noisy channel. Transmission errors can be reduced by adding redundancy to the information source. For example, a *systematic error-correcting code* [52] maps a vector of K *information bits* $u = (u_1, ..., u_K), u_i \in \{0, 1\}$, into an N-bit *codeword* $c = (u, x)$, adding $N - K$ *code bits* $x = (x_1, ..., x_{N-K}), x_j \in \{0, 1\}$. The *code rate* $R = K/N$ is the fraction of the information bits relative to the total number of transmitted bits. A broad class of systematic codes includes linear block codes. Given a binary-valued *generator matrix* G, an *(N,K) linear block code* is defined so that the codeword $c = (u, x)$ satisfies $c = uG$, assuming summation modulo 2. The bits x_i are also called the *parity-check* bits. For example, the generator matrix

```
      1 0 0 0 1 1 0
G =   0 1 0 0 1 0 1
      0 0 1 0 0 1 1
      0 0 0 1 1 1 1
```

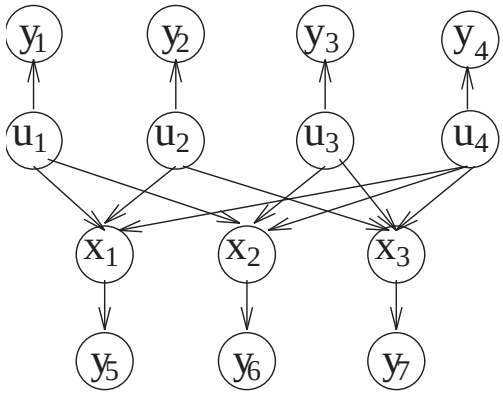


Figure 1.15: Belief network for a (7,4) Hamming code

defines a (7,4) *Hamming code*.

The codeword $c = (u, x)$, also called the *channel input*, is transmitted through a noisy channel. A commonly used Additive White Gaussian Noise (AWGN) channel model assumes that independent Gaussian noise with variance σ^2 is added to each transmitted bit, producing a *real-valued channel output* y. Given y, the decoding task is to restore the input information vector u [46, 52, 13].

It was observed that the decoding problem can be formulated as a probabilistic inference task over a belief network [52]. For example, a (7,4) Hamming code can be represented by the belief network in Figure 1.15, where the bits of u, x, and y vectors correspond to the nodes, the parent set for each node x_i is defined by non-zero entries in the $(K + i)$th column of G, and the (deterministic) conditional probability function $P(x_i|pa_i)$ equals 1 if $x_i = u_{j_1} \oplus ... \oplus u_{j_p}$ and 0 otherwise, where $\oplus$ is the summation modulo 2 (also, XOR, or parity-check operation). Each output bit y_j has exactly one parent, the corresponding channel input bit. The conditional density function $P(y_j|c_j)$ is a Gaussian (normal) distribution $N(c_j; \sigma)$, where the mean equals the value of the transmitted bit, and σ^2 is the noise variance.

The probabilistic decoding task can be formulated in two ways. Given the observed output y, the task of *bit-wise* probabilistic decoding is to find the most probable value of each *information bit*, namely:

$$u_k^* = \arg \max_{u_k \in \{0,1\}} P(u_k|y), \ for\ 1 \le k \le K.$$

The *block-wise* decoding task is to find a maximum a posteriori (maximum-likelihood) *information sequence*

$$u' = \arg \max_u P(u|y).$$

Block-wise decoding is sometimes formulated as finding a most probable explanation (MPE) assignment (u', x') to the codeword bits, namely, finding

$$(u', x') = \arg\max_{(u,x)} P(u, x|y).$$

Accordingly, bit-wise decoding, which requests the posterior probabilities for each information bit, can be solved by belief updating algorithms, while the block-wise decoding translates to the MAP or MPE tasks, respectively.

In the coding community, decoding error is measured by the *bit error rate (BER)*, defined as the average percentage of incorrectly decoded bits over multiple transmitted words (blocks). It was proven by Shannon [76] that, given the noise variance σ^2, and a fixed code rate $R = K/N$, there is a theoretical limit (called *Shannon's limit*) on the smallest achievable BER, no matter which code is used. Unfortunately, Shannon's proof is non-constructive, leaving open the problem of finding an optimal code that achieves this limit. In addition, it is known that low-error (i.e., high-performance) codes tend to be long [69], and thus intractable for exact (optimal) decoding algorithms [52]. Therefore, finding low-error codes is not enough; good codes must be also accompanied by efficient *approximate* decoding algorithms.

In recent years, several high-performance coding schemes have been proposed (*turbo codes* [4], *low-density generator matrix codes* [11], *low-density parity-check* codes [13]), that outperform by far the best up-to-date existing codes and get quite close to Shannon's limit. This was considered in its time "the most exciting and potentially important development in coding theory in many years" [52]. Surprisingly, it was observed that the decoding algorithm employed by those codes is equivalent to an iterative application of Pearl's *belief propagation* algorithm [60] that is designed for polytrees and, therefore, performs only local computations. This successful performance of iterative belief propagation (IBP) on multiply-connected coding networks suggests that approximations by local computations may be suitable for this domain. In the following section, we discuss iterative belief propagation in more detail.

As noted in Chapter **??**, iterative belief propagation (IBP) (also known as loopy belief propagation computes an approximate belief for each variable in the network. It applies Pearl's belief propagation [60], developed for singly-connected networks, to multiply-connected networks, as if there are no cycles. Nodes are processed (activated) in accordance with a variable ordering called an *activation schedule*. Processing all nodes along the given ordering yields one iteration of belief propagation. Subsequent iterations update the messages computed during previous iterations. Algorithm IBP(I) stops after I iterations. If applied to polytrees, two iterations of the algorithm are guaranteed to converge to the correct a posteriori beliefs [60]. For multiply-connected networks, however, the algorithm may not even converge, or it may converge to incorrect beliefs.

In our implementation, we assumed an activation schedule that first updates the input variables of the coding network and then updates the parity-check variables. $Bel(x)$ computed for each node can be viewed as an approximation to the posterior beliefs. The tuple generated by selecting the most probable value for each node is the output of the decoding algorithm.

We experimented with several types of (N, K) linear block codes, which include $(7, 4)$ and $(15, 11)$ Hamming codes, randomly generated codes, and structured codes with relatively low induced width. The code rate was $R = 1/2$, i.e. $N = 2K$. As described above, linear block codes can be represented by four-layer belief networks having K nodes in each layer (see Figure 1.7). The two outer layers represent the channel output $y = (y^u, y^x)$, where y^u and y^x result from transmitting the input vectors u and x, respectively. The input nodes are binary (0/1), while the output nodes are real-valued.

Random codes are generated as follows. For each parity-check bit x_j, P parents are selected randomly out of the K information bits. Random codes are similar to the *low-density generator matrix* codes [11], which randomly select a given number C of *children* nodes for each information bit.

Structured codes are generated as follows. For each parity bit x_i, P sequential parents $\{u_{(i+j) mod K}, 0 \leq j < P\}$ are selected. Figure 1.7 shows a belief network of the structured code with K=5 and P=3. Note that the induced width of the network is 3, given the order x_0,...,x_4, u_0,...,u_4. In general, a structured (N,K) block code with P parents per each code bit has induced width P, no matter how large K and N are.

Given K, P, and the channel noise variance σ^2, a coding network *instance* is generated as follows. First, the appropriate belief network structure is created. Then, an input signal is simulated, assuming uniform prior distribution of information bits. The parity-check bits are computed accordingly and the codeword is "transmitted" through the channel. As a result, Gaussian noise with variance σ^2 is added to each information and parity-check bit yielding the channel output y, namely, a real-valued assignment to the y_i^u and y_j^x nodes[5]. The decoding algorithm takes as an input the coding network and the observation (evidence) y and returns the recovered information sequence u'.

For each Hamming code network and for each structured code network, we simulate 10,000 and 1,000 input signals, respectively, and report the corresponding BERs associated with the algorithms. For the random code networks, the BER is computed over 1,000 random networks while using only one randomly generated signal per network .

The bit error rate, BER, is plotted as a function of the channel noise and is compared to the Shannon limit and to the performance of a high-quality turbo-code reported in [46]

[5]Note that simulation of the channel output is akin to the simulation of likely evidence in a general Bayesian network (i.e. forward sampling, or ancestral simulation). As observed in the previous sections, $mbe - mpe$ is more accurate, on average, when evidence is likely. Not surprisingly, similar results were observed on coding networks.

and used as a reference here (this code has a very large block size K=65,536, code rate 1/2, and was decoded using 18 iterations of IBP until convergence [46]).

In Figure 1.16, we compare the algorithms on the structured linear block code networks with N=50 and 100, K=25 and 50, and P=4 and P=7. The figures also displays the Shannon limit and the performance of IBP(18) on a state-of-the-art turbo-code having input block size K=65,536 and rate 1/2 (the results are copied from [46]). Clearly, our codes are far from being optimal: even the exact *elim-mpe* decoding yields a much higher error than the turbo-code. However, the emphasis of our preliminary experiments was not on improving the state-of-the-art decoder but rather on evaluating the performance of *mbe-mpe* and comparing it to the performance of IBP(i) and *mbe-mpe* on different types of networks.

The experiments showed that

1. On a class of structured codes having low induced width the mini-bucket approximation *mbe-mpe* outperforms IBP;

2. On a class of random networks having large induced width and on some Hamming codes IBP outperforms *mbe-mpe*;

3. As expected, the exact MPE decoding (*elim-mpe*) outperforms approximate decoding. However, on random networks, finding exact MPE was not feasible due to the large induced width.

1.9 Mini-Clustering

Clearly the mini-bucket idea can be extended to any tree-decomposition scheme. In this section we will describe such an extension called *Mini-Clustering (MC)* to general tree decompositions. 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. The resulting approximation scheme allows adjustable levels of accuracy and efficiency, in anytime style.

We will describe mini-clustering relative to a unifying tree-decomposition framework which generalizes tree-decompositions to include join-trees, bucket-trees and other variants applicable to both constraint processing and probabilistic inference.

Definition 1.9.1 (tree-decomposition, cluster tree) *Let $BN =< X, D, G, P >$ be a belief network. A* tree-decomposition *for BN is 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:*

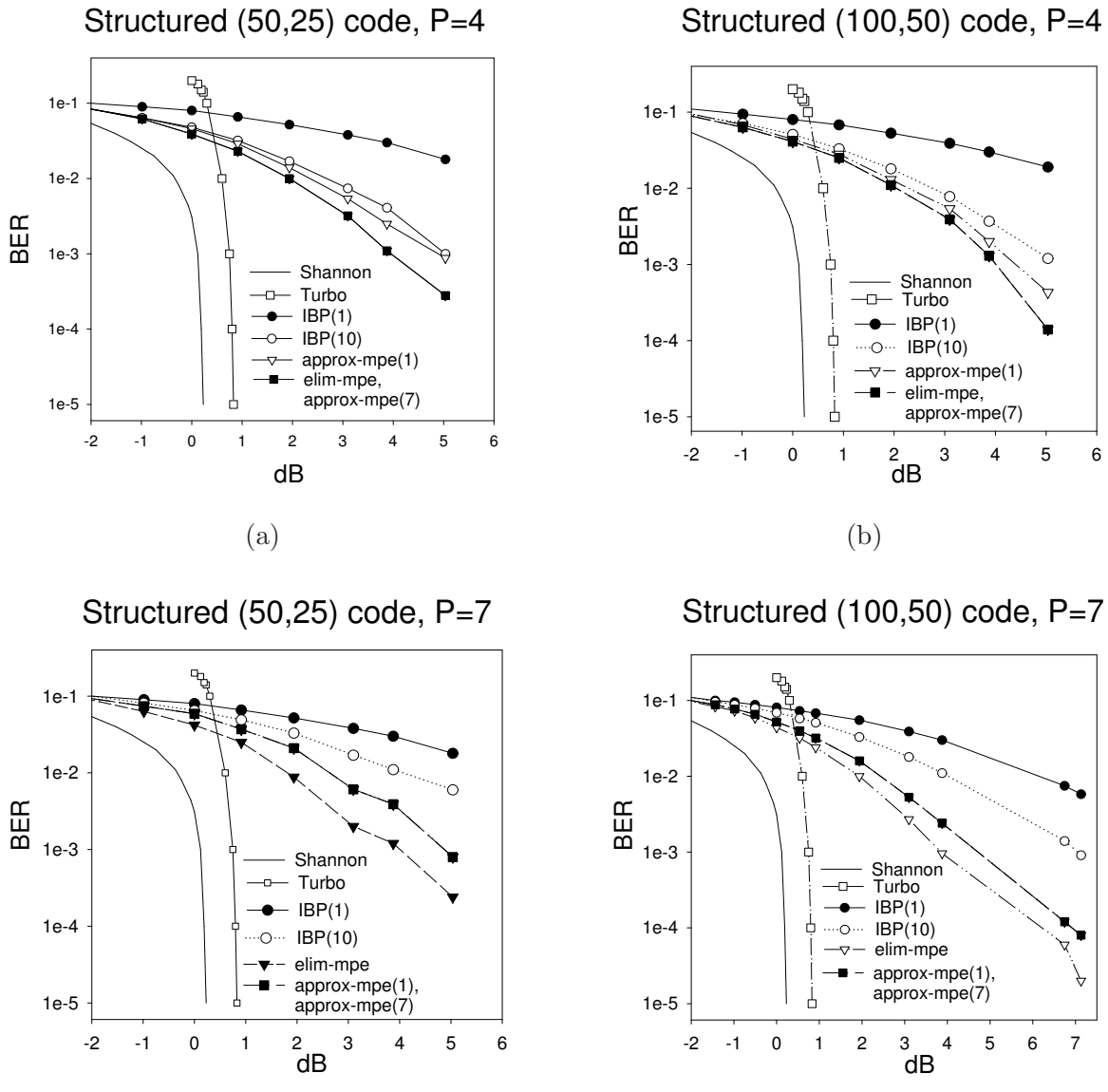
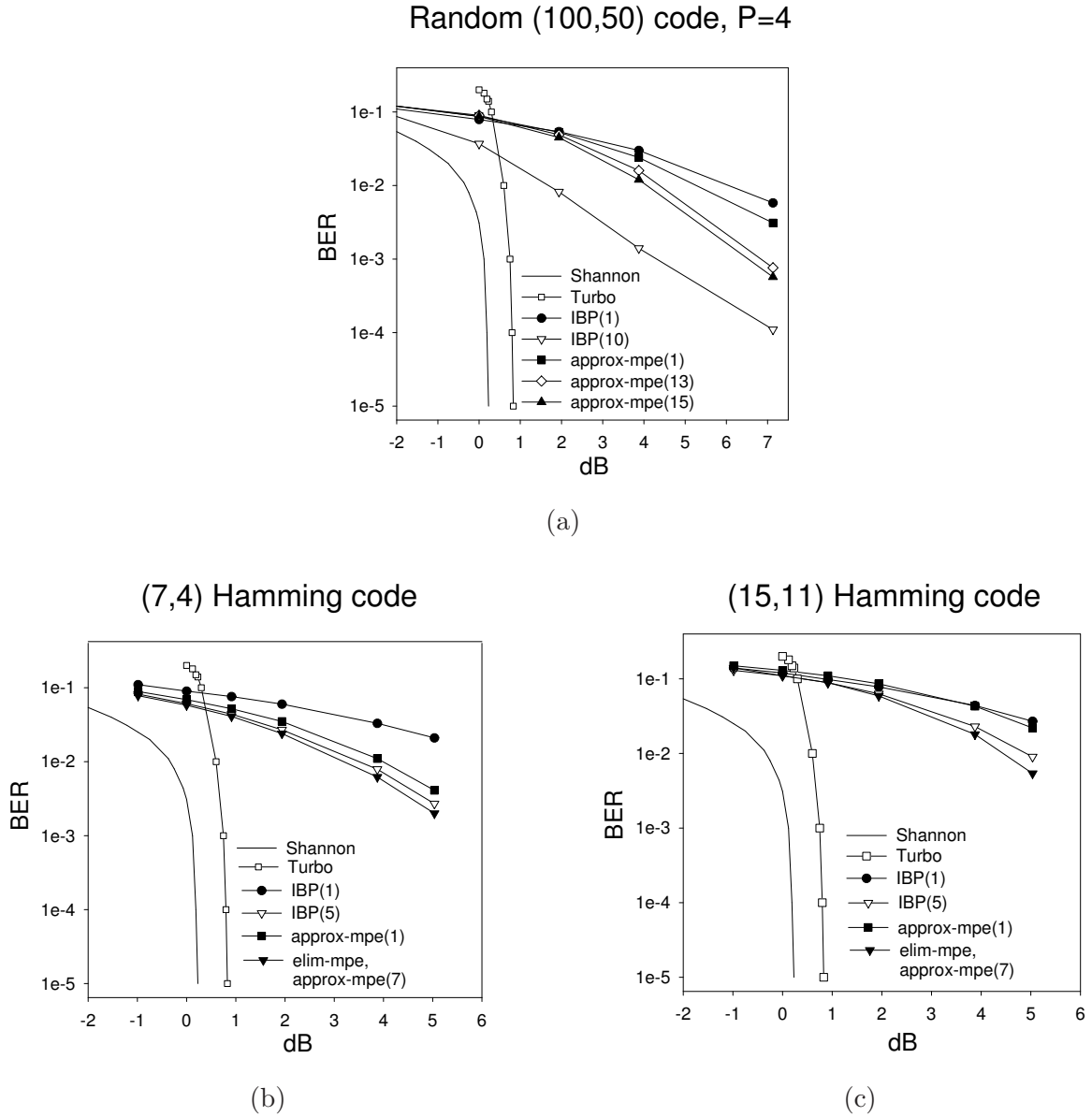


Figure 1.16: The average performance of *elim-mpe*, *mbe-mpe*(i), and IBP(I) on rate 1/2 **structured block codes** and 1000 randomly generated input signals. The induced width of the networks is: (a),(b) $w^* = 6$; (c),(d) $w^* = 12$. The bit error rate (BER) is plotted versus the channel noise measured in decibels (dB), and compared to the Shannon limit and to the performance of IBP(18) on a high-quality code reported [46] (a turbo-code having input block size K=65,536 and rate 1/2). Notice that *mbe-mpe*(7) coincides with *elim-mpe* in (a) and (b), while in (c) and (d) it coincides with *mbe-mpe*(1).



(a)

(b) (c)

Figure 1.17: The average performance of *elim-mpe*, *mbe-mpe*(i), and IBP(I) on (a) 1000 instances of rate 1/2 **random block codes**, one signal instance per code; and on (b) (7,4) and (c) (15,11) **Hamming codes**, 1000 signal instances per each code. The induced width of the networks is: (a) $30 \leq w^* \leq 45$; (b) $w^* = 3$; (c) $w^* = 9$. The bit error rate (BER) is plotted versus the channel noise measured in decibels (dB), and compared to the Shannon limit and to the performance of IBP(18) on a high-quality code reported [46] (a turbo-code having input block size K=65,536 and rate 1/2). Notice that *mbe-mpe*(7) coincides with *elim-mpe* in (a) and (b), while in (c) and (d) it coincides with *mbe-mpe*(1).

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. *For each variable* $X_i \in X$, *the set* $\{v \in V | X_i \in \chi(v)\}$ *induces a connected subtree of* T. *This is also called the running intersection property.*

We will often refer to a node and its functions as a cluster *and use the term* tree-decomposition *and* cluster tree *interchangeably.*

Definition 1.9.2 (treewidth, hypertreewidth, separator-width, eliminator) *The* treewidth *[2] of a tree-decomposition* $< T, \chi, \psi >$ *is* $max_{v \in V} |\chi(v)|$, *and its* hypertreewidth *is* $max_{v \in V} |\psi(v)|$. *Given two adjacent vertices* u *and* v *of a tree-decomposition, the* separator *of* u *and* v *is defined as* $sep(u, v) = \chi(u) \cap \chi(v)$, *and the* eliminator *of* u *with respect to* v *is* $elim(u, v) = \chi(u) - \chi(v)$. *The separator-width is the maximum over all separators.*

Join-Trees and Cluster Tree Elimination

In both Bayesian network and constraint satisfaction communities, the most used tree decomposition method is called join-tree decomposition [50, 25] (also called junction-trees). Such decompositions can be generated by embedding the network's moral graph, G, in a chordal graph, often using a triangulation algorithm and using its maximal cliques as nodes in the join-tree. The triangulation algorithm assembles a join-tree by connecting the maximal cliques in the chordal graph in a tree. Subsequently, every CPT p_i is placed in one clique containing its scope. A join-tree decomposition of a belief network (G, P) is a tree $T = (V, E)$, where V is the set of cliques of a chordal graph G' that contains G, and E is a set of edges that form a tree between cliques, satisfying the running intersection property [51].

There are a few variants for processing join-trees for belief updating [38, 75]. The variant which we use here, is cluster-tree-elimination (CTE) (described in Chapter 2), is applicable to any tree-decompositions and is geared toward space savings. It is a message passing algorithm (either two-phase message passing, or in asynchronous mode), where messages are computed by summation (or any combine operator) over the eliminator between the two clusters of the product of functions in the originating cluster. Algorithm CTE for belief updating denoted CTE-BU is described again in Figure 1.19. The algorithm pays a special attention to the processing of observed variables since the presence of evidence is a central component in belief updating. When a cluster sends a message to a neighbor, the algorithm operates on all the functions in the cluster except the message from that particular neighbor. The message contains a single *combined* function and *individual* functions that do not share variables with the relevant eliminator. All the non-individual functions are *combined* in a product and summed over the eliminator.

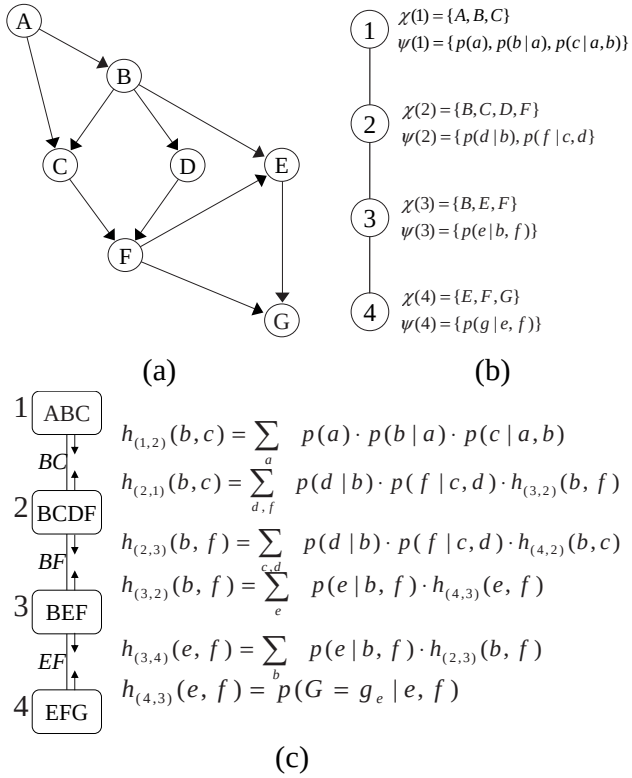


Figure 1.18: *a*) A belief network; *b*) A join-tree decomposition; *c*)Execution of CTE-BU; no individual functions appear in this case

Example 1.9.3 Figure 1.18 describes a belief network (a) and a join-tree decomposition for it (b). Figure 1.18c shows the trace of running CTE-BU. In this case no individual functions appear between any of the clusters. To keep the figure simple, we only show the combined functions $h_{(u,v)}$ (each of them being in fact the only element of the set $H_{(u,v)}$ that represents the corresponding message between clusters u and v). □

Theorem 1.9.4 (Complexity of CTE-BU) *The time complexity of CTE-BU is* $O(deg \cdot (n+N) \cdot d^{w^*+1})$ *and the space complexity is* $O(N \cdot d^{sep})$*, where deg is the maximum degree of a node in the tree, n is the number of variables, N is the number of nodes in the tree decomposition, d is the maximum domain size of a variable,* w^* *is the treewidth and sep is the maximum separator size.*

Proof: The number of cliques in the chordal graph G' corresponding to G is at most n, so the number of nodes in the join-tree is at most n. The complexity of processing a node

Algorithm **CTE for Belief-Updating (CTE-BU)**
Input: A tree decomposition $< T, \chi, \psi >$, $T = (V, E)$ for $BN =< X, D, G, P >$. Evidence variables $var(e)$.
Output: An augmented tree whose nodes are clusters containing the original CPTs and the messages received from neighbors. $P(X_i, e)$, $\forall X_i \in X$.

Denote by $H_{(u,v)}$ the message from vertex u to v, $ne_v(u)$ the neighbors of u in T 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.

• **Compute messages:**
For every node u in 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 function:**

$$h_{(u,v)} = \sum_{elim(u,v)} \prod_{f \in A} f.$$

   where A is the set of functions in $cluster_v(u)$ whose scope intersects $elim(u, v)$.
   Add $h_{(u,v)}$ to $H_{(u,v)}$ and add all the individual functions in $cluster_v(u) - A$
   Send $H_{(u,v)}$ to node v.

• **Compute $P(X_i, e)$:**
For every $X_i \in X$ let u be a vertex in T such that $X_i \in \chi(u)$. Compute $P(X_i, e) = \sum_{\chi(u)-\{X_i\}} (\prod_{f \in cluster(u)} f)$

Figure 1.19: Algorithm Cluster-Tree-Elimination for Belief Updating (CTE-BU)

u in the join-tree is $deg_u \cdot (|\psi(u)| + deg_u - 1) \cdot exp(|\chi(u)|)$, where deg_u is the degree of u. By bounding deg_u by deg and $\chi(u)$ by w^*, and summing over all nodes, we can bound the entire time complexity by $O(deg \cdot n \cdot exp(w))$.

For each edge JTC records functions. Since the number of edges in bounded by n and the size of each message is bounded by $exp(sep)$ we get space complexity of $O(n \cdot exp(sep))$.
□

1.9.1 Mini-Clustering for Belief Updating

The time, and especially the space complexity of CTE-BU renders the algorithm infeasible for problems with large treewidth, and it seems natural extension of the mini-bucket idea to tree-decompositions. Indeed, rather than computing the mini-bucket approximation n times, one for each variable as would be required by the mini-bucket approach, the algorithm performs an equivalent computation with just two message passings along each arc of the cluster tree. The idea is to partition each cluster into mini-clusters having at most i variables, where i is an accuracy parameter. (We could also use number of functions m as a bounding parameter similar to the mini-bucket case.) Node u partitions its cluster into p mini-clusters $mc(1), \ldots, mc(p)$. Instead of computing $h_{(u,v)} = \sum_{elim(u,v)} \prod_{k=1}^{p} \prod_{f \in mc(k)} f$ as in CTE-BU, we can compute an upper bound by migrating the summation operator into each mini-cluster. However, this would give $\prod_{k=1}^{p} \sum_{elim(u,v)} \prod_{f \in mc(k)} f$ which is an unnecessarily large upper bound on $h_{(u,v)}$ in which each $\prod_{f \in mc(k)} f$ is bounded by its sum over $elim(u,v)$. Instead, we rewrite $h_{(u,v)} = \sum_{elim(u,v)} (\prod_{f \in mc(1)} f) \cdot (\prod_{i=2}^{p} \prod_{f \in mc(i)} f)$. Subsequently, instead of bounding $\prod_{f \in mc(i)} f, (i \geq 2)$ by summation over the eliminator, we bound it by its maximum over the eliminator, which yields $(\sum_{elim(u,v)} \prod_{f \in mc(1)} f) \cdot \prod_{k=2}^{p} (\max_{elim(u,v)} \prod_{f \in mc(k)} f)$. Therefore, if we are interested in an upper bound, we marginalize one mini-cluster by summation and the others by maximization. Note that the summation in the first mini-cluster must be over *all* variables in the eliminator, even if some of them might not appear in the scope of functions in $mc(1)$.

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. One of the mini-clusters is processed by summation and the others by maximization 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 (see above derivation for rationale). 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 (and indeed the empirical evaluation encourages that) 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 1.20.

Example 1.9.5 Figure 1.21 shows the trace of running MC-BU(3) on the problem in

Procedure **MC for Belief Updating (MC-BU(i))**

2. **Compute the combined mini-functions:**

   Make an (i)-size mini-clusters partitioning of $cluster_v(u)$, $\{mc(1), \ldots, mc(p)\}$;

   $h^1_{(u,v)} = \sum_{elim(u,v)} \prod_{f \in mc(1)} f$

   $h^i_{(u,v)} = \max_{elim(u,v)} \prod_{f \in mc(i)} f \quad i = 2, \ldots, p$

   add $\{h^i_{(u,v)} | i = 1, \ldots, p\}$ to $H_{(u,v)}$. Send $H_{(u,v)}$ to v.

Compute upper bounds 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
$(\sum_{\chi(u)-X_i} \prod_{f \in mc(1)} f) \cdot (\prod_{k=2}^{p} \max_{\chi(u)-X_i} \prod_{f \in mc(k)} f)$.

Figure 1.20: Procedure Mini-Clustering for Belief Updating (MC-BU)

Figure 1.18. First, evidence $G = g_e$ is assigned in all CPTs. There are no individual functions to be sent from cluster 1 to cluster 2. Cluster 1 contains only 3 variables, $\chi(1) = \{A, B, C\}$, therefore it is not partitioned. The combined function $h^1_{(1,2)}(b, c) = \sum_a p(a) \cdot p(b|a) \cdot p(c|a, b)$ is computed and the message $H_{(1,2)} = \{h^1_{(1,2)}(b, c)\}$ is sent to node 2. Now, node 2 can send its message to node 3. Again, there are no individual functions. Cluster 2 contains 4 variables, $\chi(2) = \{B, C, D, F\}$, and a partitioning is necessary: MC-BU(3) can choose $mc(1) = \{p(d|b), h_{(1,2)}(b, c)\}$ and $mc(2) = \{p(f|c, d)\}$. The combined functions $h^1_{(2,3)}(b) = \sum_{c,d} p(d|b) \cdot h_{(1,2)}(b, c)$ and $h^2_{(2,3)}(f) = \max_{c,d} p(f|c, d)$ are computed and the message $H_{(4,3)} = \{h^1_{(2,3)}(b), h^2_{(2,3)}(f)\}$ is sent to node 3. The algorithm continues until every node has received messages from all its neighbors. An upper bound on $p(a, G = g_e)$ can now be computed by choosing cluster 1, which contains variable A. It doesn't need partitioning, so the algorithm just computes $\sum_{b,c} p(a) \cdot p(b|a) \cdot p(c|a, b) \cdot h^1_{(2,1)}(b) \cdot h^2_{(2,1)}(c)$. Notice that unlike CTE-BU which processes 4 variables in cluster 2, MC-BU(3) never processes more than 3 variables at a time. □

Theorem 1.9.6 *Algorithm MC-BU(i) with max (respectively min) computes an upper (respectively lower) bound on the joint probability $P(X, e)$ of each variable and each of its values.*

Theorem 1.9.7 (Complexity of MC-BU(i)) *[43] 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*

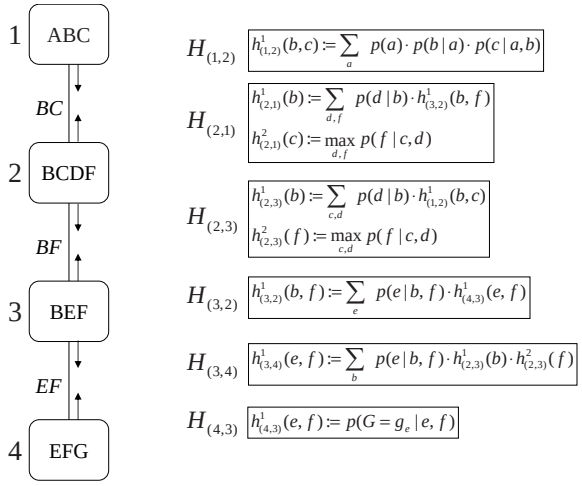


Figure 1.21: Execution of MC-BU for $i = 3$

size of a variable and $hw^* = max_u|\{f|scope(f) \cap \chi(u) \neq \varphi\}|$*, which bounds the number of functions that may travel to a neighboring cluster via message-passing.*

Accuracy. For a given i, the accuracy of MC-BU(i) is not worse than that of executing the mini-bucket algorithm MB(i) n times, once for each variable (an algorithm that we call nMB(i)). Given a specific execution of MC-BU(i), we can show that for every variable X_i, there exists an ordering of the variables and a corresponding partitioning such that MB(i) computes the same approximation value for $P(X_i, e)$ as does $MC - BU(i)$. In empirical analysis [40] it is shown that MC-BU has an up to linear speed-up over nMB(i).

The MC-BU algorithm using *max* operator computes an upper bound $\overline{P}(X_i, e)$ on the joint probability $P(X_i, e)$. However, as seen in the case of mini-bucket, 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 $P(X_i|e)$. In principle, if we can derive a lower bound $\underline{P}(e)$ on $P(e)$, then we can compute $\overline{P}(X_i, e)/\underline{P}(e)$ as an upper bound on $P(X_i|e)$. However, due to compound error, it is likely to be ineffective. In the empirical evaluation we experimented with normalizing the upper bound as $\overline{P}(X_i, e)/\sum_{X_i} \overline{P}(X_i, e)$ over the values of X_i. The result is not necessarily an upper bound on P($X_i|e$). Similarly, we can also normalize the values when using *mean* or *min* operators. 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.

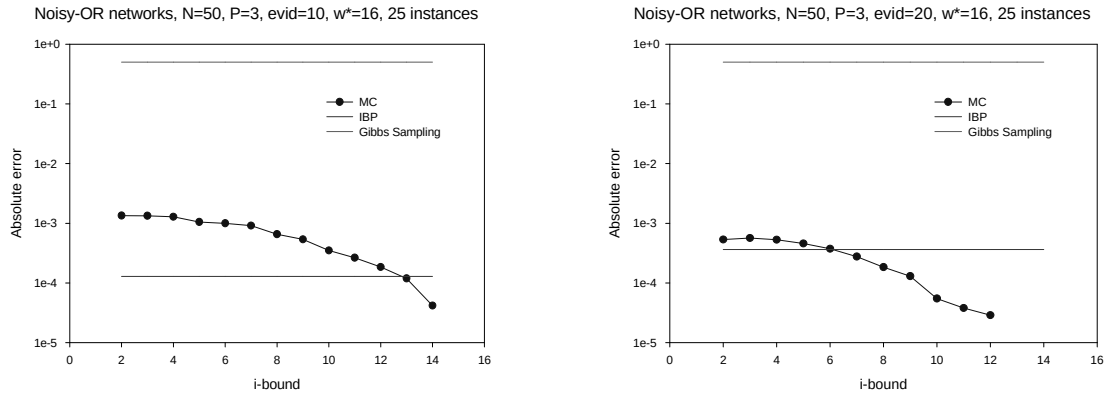


Figure 1.22: Absolute error for noisy-OR networks

1.9.2 Empirical Evaluation

When evaluating MC-BU empirically it was immediately observed that the quality of MC-BU in providing upper or lower bounds on the joint $P(X_i, e)$ was ineffective. Although the upper bound decreases as the accuracy parameter i increases, it still is in many cases greater than 1. Therefore, following the ideas explained earlier we report the results with normalizing the upper bounds (called max) and normalizing the mean (called $mean$). We notice that MC-BU using the $mean$ operator is doing consistently better.

In additional to the absolute and ratio accuracy measures we use Normalized Hamming Distance (NHD) - We picked the most likely value for each variable for the approximate and for the exact, took the ratio between the number of disagreements and the total number of variables, and averaged over the number of problems that we ran for each class. We show some of the main results here. For more details see [66].

We have experimented with random grid networks. The grid networks have the structure of a square, with edges directed to form a diagonal flow (all parallel edges have the same direction). They were generated by specifying N (a square integer) and K (we used K=2).

Random noisy-or networks results are summarized in Figure 1.22. For NHD, both IBP and MC-BU gave perfect results. For the other measures, we noticed that IBP is more accurate for no evidence by about an order of magnitude. However, as evidence is added, IBP's accuracy decreases, while MC-BU's increases and they give similar results. Clearly, MC-BU gets better as the accuracy parameter i increases, which shows its anytime behavior.

General random networks results are summarized in Figure 1.23. They are in general similar to those for random noisy-or networks. NHD is non-zero in this case. Again, IBP has the best result only for few evidence variables. It is remarkable how quickly MC-BU

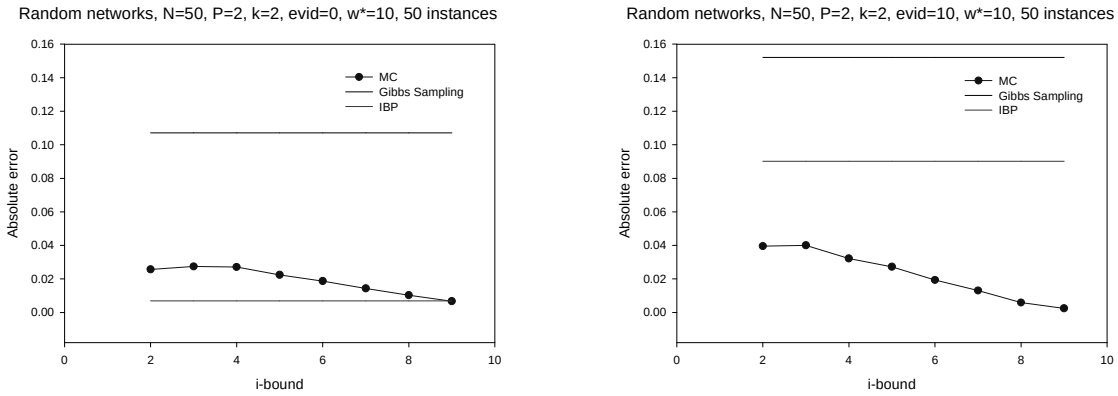


Figure 1.23: Absolute error for random networks

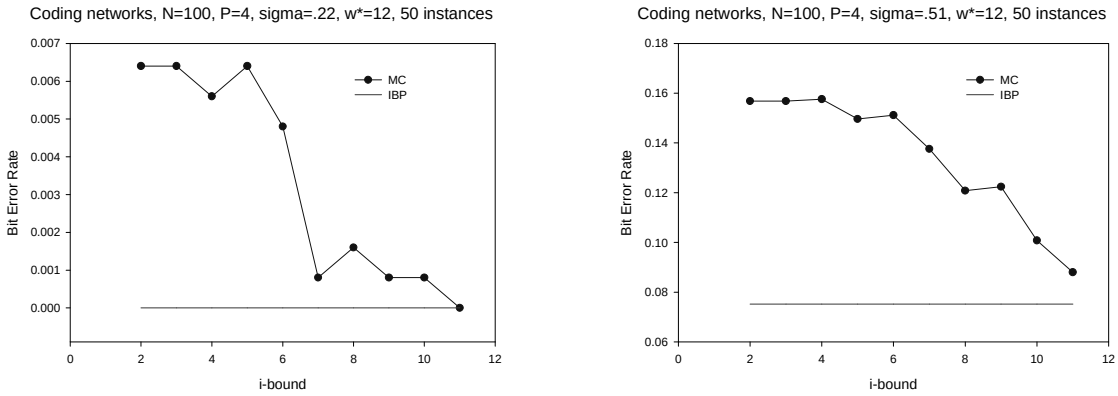


Figure 1.24: BER for coding networks

surpasses the performance of IBP as evidence is added.

Random coding networks results are given in Table Figure 1.24. The instances fall within the class of linear block codes, (σ is the channel noise level). These are the only problems that we experimented with where IBP outperformed MC-BU throughout.

Grid networks results are given in Figure **??**. We only report results with *mean* operator for a 15x15 grid for which the induced width is w*=22. We notice that IBP is more accurate for no evidence and MC is better as more evidence is added. The same behavior was consistently manifested for smaller grid networks (from 7x7 up to 14x14).

CPCS networks results. We also tested on three CPCS benchmark files. The results are given Figure 1.25. It is interesting to notice that the MC scheme scales up even to fairly large networks, like the real life example of CPCS422 (induced width 23). IBP is again slightly better for no evidence, but is quickly surpassed by MC when evidence is added.

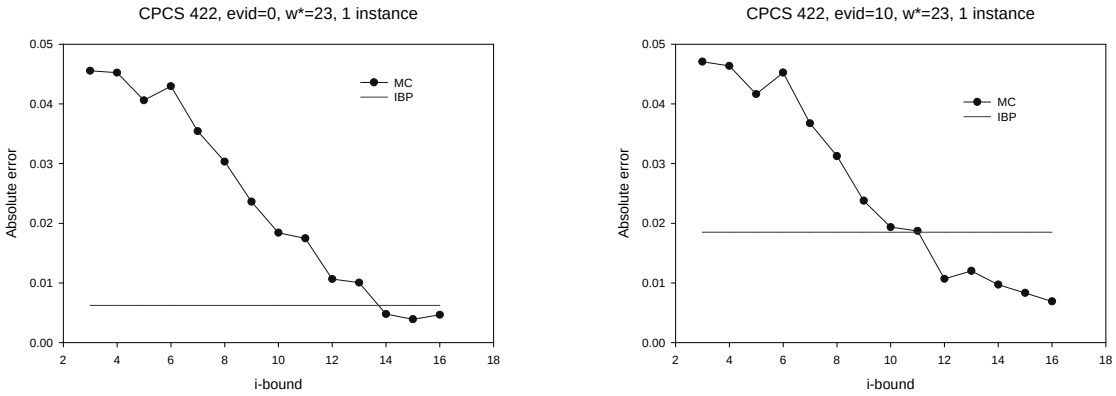


Figure 1.25: Absolute error for CPCS422

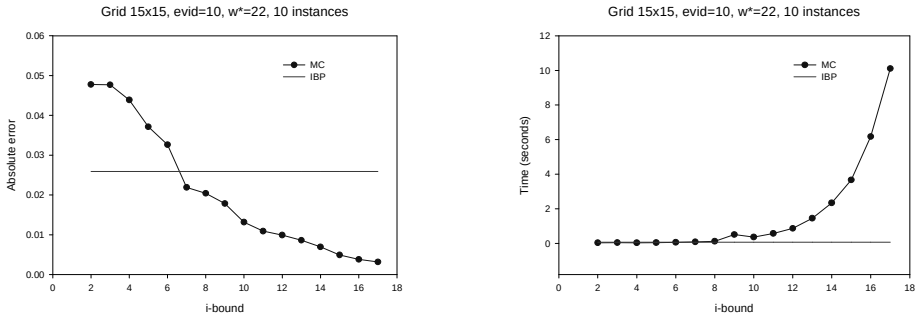


Figure 1.26: Absolute error and time for grid networks

There are many potential ways for improving the MC scheme. Among the most important, the partitioning step can be further elaborated. One extension is called Iterative Join-Graph Propagation (IJGP), which belongs to the class of generalized belief propagation methods [37].

1.10 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 the accuracy. The immediate question is if we can combine the anytime property of MC 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.

Definition 1.10.1 (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. *The connectedness requirement is also called the running intersection property.*

We will refer to a node and its CPT functions as a *cluster*[6] 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.

Join-Tree Propagation. The well known join-tree clustering algorithm first converts the belief network into a cluster tree and then sends messages between clusters. We call the second message passing phase *join-tree propagation*. The complexity of join-tree clustering is exponential in the number of variables in a cluster (treewidth), and the number of variables in the intersections between adjacent clusters (separator-width), as defined below.

Definition 1.10.2 (arc-minimality) *A join-graph decomposition* D *is* arc-minimal *if none of its arcs can be removed while still satisfying the connectedness property of Definition 1.10.1.*

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.

[6]Note that a node may be associated with an empty set of CPTs

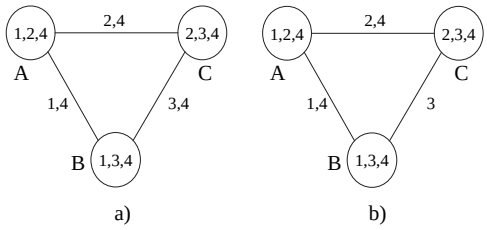


Figure 1.27: An arc-labeled decomposition

Example 1.10.3 The example in Figure 1.27a 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. □

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 1.10.4 ((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.*

Definition 1.10.5 (separator, eliminator) *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 minimal by deleting variables from the labels. It is easy to see that a *minimal arc-labeled join-graph* does not contain any cycle relative to any single variable. That is, any two clusters containing the same variable are connected by exactly one path labeled with that variable.

The dual join-graph. The dual join-graph is an interesting special case where each node contains a single function.

It is easy to see that a dual graph is a join-graph. Borrowing the notion of acyclicity from relational databases, we define a belief network to be acyclic if its dual graph has a join-tree, namely if it has an arc-minimal version that is a tree. Clearly, when join-tree clustering is applied to such a join-tree it will converge to the correct values. Note that while polytrees are a special case of acyclic belief networks, acyclic networks include more than just polytrees. For example, the simple network that contains $\{P(A), P(B|A), P(C|A,B)\}$ is acyclic. The tree connecting $P(A)$ to $P(B|A)$ and $P(B|A)$ to $P(C|B,A)$ is a join-tree. When join-tree clustering is applied to this network along an arc-minimal dual graph decomposition, it is guaranteed to compute the exact values. On the other hand, if belief propagation is applied to this network, it will neither necessarily converge nor be exact. While this observation is a side issue, it is worthwhile summarizing,

Proposition 1.10.6 *Algorithm join-tree propagation is tractable and exact for every acyclic belief network.*

If the network is not acyclic, join-tree propagation can be applied to an arc-minimal dual decomposition, iteratively. More generally, we can consider applying join-tree propagation iteratively, to join-graphs decompositions whose clusters may include more than one CPT. This leads to the definition of join-tree propagation over join-graphs, an algorithm which we call *Iterative Join-Graph Propagation (IJGP).*

1.10.1 Algorithm IJGP

Applying CTE propagation iteratively to join-graphs yields algorithm *Iterative Join-Graph Propagation (IJGP)* described in Figure 1.28. One iteration of the algorithm applies message-passing in a topological order over the join-graph, forward and back.

When node i sends a message (or messages) to a neighbor node j it operates on all the CPTs in its cluster and on all the messages sent from its neighbors excluding the ones received from j. First, all individual functions that share no variables with the eliminator are collected and sent to j. All the rest of the functions are *combined* in a product and summed over the eliminator between i and j.

Clearly

Theorem 1.10.7
1. *[50] If IJGP is applied to a join-tree decomposition it reduces to CTE and it therefore is guaranteed to compute the exact beliefs in one iteration.*
2. *[47] 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*

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

Input An arc-labeled join-graph decomposition $< JG, \chi, \psi, \theta >, JG = (V, E)$ for $BN =< X, D, G, P >$. 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 $p_i \in \psi(u)$ $\chi(u) := \chi(u) - var(e),\ \ \forall u \in V$
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 T 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 1.28: Algorithm Iterative Join-Graph Propagation (IJGP)

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.

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

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

Note however that when applied to a join-graph the algorithm is neither guaranteed to converge nor to find the exact posterior.

1.10.2 I-Mappness of Arc-Labeled Join-Graphs

The effectiveness of IJGP, no doubt, will depend on the choice of cluster graphs it operates on. The following paragraphs provide some rationale to our choice of minimal arc-labeled join-graphs. First, we are committed to the use of an underlying graph structure that captures as many of the distribution independence relations as possible, without introducing new ones. That is, we restrict attention to cluster graphs that are I-maps of P [59]. Second, we wish to avoid cycles as much as possible in order to minimize computational over-counting.

It can be shown that any join-graph of a belief network is an I-map of the underlying probability distribution relative to node-separation. It turns out that arc-labeled join-graphs display a richer set of independencies relative to arc-separation.

Definition 1.10.8 (arc-separation in (arc-labeled) join-graphs) *Let $D =< JG, \chi, \psi, \theta >$, $JG = (V, E)$ be an arc-labeled decomposition. 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 ($W = \cup_{v \in N_W} \chi(v)$, $Z = \cup_{e \in E_Z} \theta(e)$). E_Z* arc-separates *N_W and N_Y in D if there is no path between N_W and N_Y in the graph JG with the edges in E_Z removed. In this case we also say that W is* separated *from Y given Z in D, and write $< W|Z|Y >_D$. Arc-separation in a regular join-graph is defined relative to its separators.*

Theorem 1.10.9 *Any arc-labeled join-graph decomposition $D =< JG, \chi, \psi, \theta >$ of a belief network $BN =< X, D, G, P >$ is an I-map of P relative to arc-separation.*

Proof: Exercise ∎

Interestingly however, removing arcs or labels from arc-labeled join-graphs whose clusters are fixed will not increase the independencies captured by arc-labeled join-graphs.

Theorem 1.10.10 *Any arc-labeled join-graph decomposition of a belief network $BN =< X, D, G, P >$ is a minimal I-map of P relative to arc-separation.*

Hence, the issue of minimizing computational over-counting due to cycles appears to be orthogonal to maximizing independencies via minimal I-mappness. Nevertheless, to avoid over-counting as much as possible, we still prefer join-graphs that minimize cycles relative to each variable. That is, we prefer to apply IJGP to *minimal* arc-labeled join-graphs.

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 arcs 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 1.29: Algorithm Join-Graph Structuring(i)

1.10.3 Generating Bounded Join-Graphs

Since we want to control the complexity of IJGP 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 one full iteration of IJGP(i) is exponential in i. How can good graph-decompositions of bounded cluster size be generated?

Since we want the join-graph to be as close as possible to a tree, and since a tree has a treewidth 1, we may try to find a join-graph JG of bounded cluster size whose treewidth (as a graph) is minimized. While we will not attempt to optimally solve this task, we will present one method for generating i-bounded graph-decompositions. The question of finding a minimal "outer treewidth" decomposition is obviously hard and should be investigated.

Definition 1.10.11 (external and internal widths) *Given an arc-labeled join-graph decomposition $D =< JG, \chi, \psi, \theta >$ of a network $< G, P >$, the internal width of D is $max_{v \in V}|\chi(v)|$, while the external width of D is the treewidth of JG as a graph.*

Clearly, if D is a tree-decomposition its external width is 1 and its internal width equals its treewidth. For example, an edge minimal dual decomposition has an internal width equal to the maximum scope of each function, m, and external width w^* which is the treewidth of the moral graph of G. On the other hand, a tree-decomposition has internal width of w^* and external width of 1.

Using this terminology we can now state the desired decompositions more clearly. Given a graph G, and a bounding parameter i we wish to find a join-graph decomposition of G whose internal width is bounded by i and whose external width is minimized. The bound i controls the complexity of one iteration of $IJGP$ while the external width provides some intuitive measure of its accuracy.

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 arc between mb and the mini-bucket (created later) of f.

Figure 1.30: Procedure Schematic Mini-Bucket(i)

Example 1.10.12 Consider our original example. Figure 1.31c shows a join-graph decomposition based on the dual graph and 1.31b just another join-graph decomposition. A join-tree decomposition was given in Figure 1.18b. The cluster sizes of decompositions 1.31b and 1.31c are 3 while their width is 2. The cluster size of the tree-decomposition is 4 and its width is 1. □

The implied optimization task is: given a constant i, find an arc-labeled join-graph decomposition whose cluster size is bounded by i such that its join-graph has a minimal induced width.

One class of such decompositions is partition-based. It starts from a given tree-decomposition of the network and then partitions the clusters until the decomposition has clusters bounded by i. The problem we get can be characterize by duplicating a variable for each partition as we have seen before. The opposite approach is grouping-based. It starts from an arc-minimal dual-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 we should attempt to reduce the external treewidth of the generated graph-decomposition. Jon-Graph structuring is a partition-based approach inspired by the mini-bucket idea.

Given a bound i, algorithm *join-graph structuring(i)* applies procedure *schematic mini-bucket(i)*, described in Figure 1.30. The procedure only traces the scopes of the functions that would be generated by the full mini-bucket procedure, avoiding actual computation. The algorithm then connects the mini-buckets' scopes minimally to obtain the running intersection property, as described in Figure 1.29.

Example 1.10.13 Figure 1.31a shows the trace of procedure schematic mini-bucket(3)

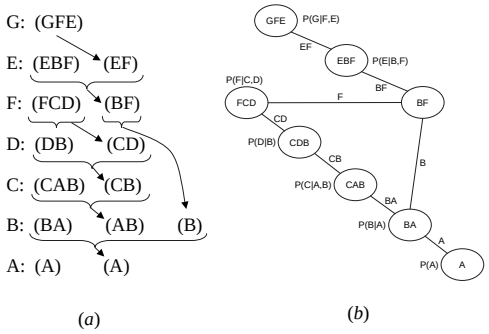


Figure 1.31: Join-graph decompositions

applied to the problem described in Figure 1.18a. The decomposition in Figure 1.31b 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. □

Procedure schematic mini-bucket ends with a collection of trees rooted in mini-buckets of the first variable. Each of these trees is minimally arc-labeled. Then, *in-edges* are labeled with only one variable, and they are added only to obtain the running intersection property between branches of these trees. It can be shown that:

Proposition 1.10.14 *Algorithm join-graph structuring(i), generates a minimal arc-labeled join-graph decomposition having bound i.*

Example 1.10.15 Figure 1.32 shows a range of arc-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 IJGP 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. □

MC(i) vs. IJGP(i). As can be hinted by our structuring of a bounded join-graph, there is a close relationship between MC(i) and IJGP(i). In particular, one iteration of IJGP(i) is similar to MC(i) (MC(i) is an algorithm that approximates join-tree clustering and was shown to be competitive with IBP and Gibbs Sampling [67]). Indeed, while we view IJGP(i) as an iterative version of MC(i), the two algorithms differ in several

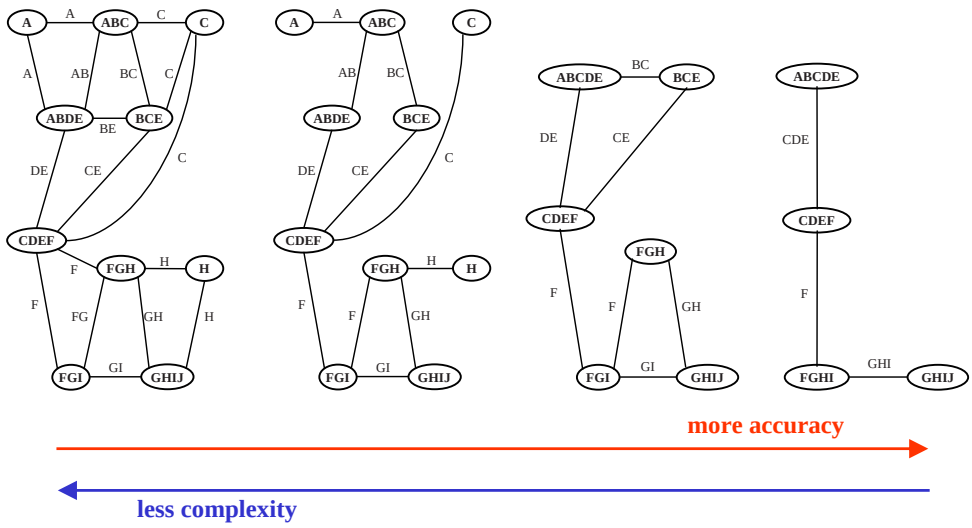


Figure 1.32: Join-graphs

technical points, some may be superficial, due to implementation, others may be more principled. We will leave the discussion at that and will observe the comparison of the two approaches in the empirical section.

1.10.4 Empirical Evaluation

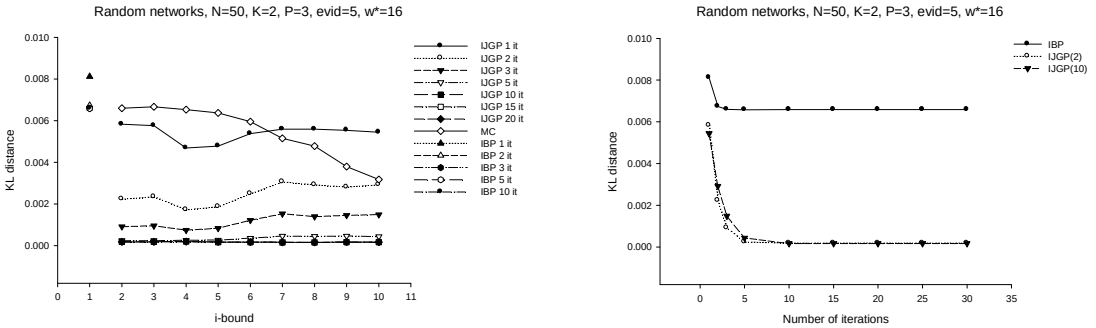
We tested the performance of IJGP(i) on random networks, on M-by-M grids, on two benchmark CPCS files with 54 and 360 variables, respectively (these are belief networks for medicine, derived from the Computer based Patient Case Simulation system, known to be hard for belief updating) and on coding networks. On each type of networks, we ran Iterative Belief Propagation (IBP), MC(i) and IJGP(i), while giving IBP and IJGP(i) the same number of iterations.

We use the partitioning method described in Section 1.10.3 to construct a join-graph. To determine the order of message computation, we recursively pick an edge (u,v), such that node u has the fewest incoming messages missing.

For each network except coding, we compute the exact solution and compare the accuracy of algorithms using: 1. Absolute error - the absolute value of the difference between the approximate and the exact, averaged over all values, all variables and all problems. 2. Relative error - the absolute value of the difference between the approximate and the exact, divided by the exact, averaged over all values, all variables and all problems. 3. KL (Kullback-Leibler) distance - $P_{exact}(X = a) \cdot log(P_{exact}(X = a)/P_{approximation}(X =$

Table 1.1: Random networks: N=50, K=2, C=45, P=3, 100 instances, w*=16

		Absolute error				Relative error				KL distance				
		IBP	IJGP			IBP	IJGP			IBP	IJGP			IBP
#it	#evid		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8	
1	0	0.02988	0.03055	0.02623	0.02940	0.06388	0.15694	0.05677	0.07153	0.00213	0.00391	0.00208	0.00277	0.0017
	5	0.06178	0.04434	0.04201	0.04554	0.15005	0.12340	0.12056	0.11154	0.00812	0.00582	0.00478	0.00558	0.0013
	10	0.08762	0.05777	0.05409	0.05910	0.23777	0.18071	0.14278	0.15686	0.01547	0.00915	0.00768	0.00899	0.0013
5	0	0.00829	0.00636	0.00592	0.00669	0.01726	0.01326	0.01239	0.01398	0.00021	0.00014	0.00015	0.00018	0.0066
	5	0.05182	0.00886	0.00886	0.01123	0.12589	0.01967	0.01965	0.02494	0.00658	0.00024	0.00026	0.00044	0.0060
	10	0.08039	0.01155	0.01073	0.01399	0.21781	0.03014	0.02553	0.03279	0.01382	0.00055	0.00042	0.00073	0.0048
10	0	0.00828	0.00584	0.00514	0.00495	0.01725	0.01216	0.01069	0.01030	0.00021	0.00012	0.00010	0.00010	0.0130
	5	0.05182	0.00774	0.00732	0.00708	0.12590	0.01727	0.01628	0.01575	0.00658	0.00018	0.00017	0.00016	0.0121
	10	0.08040	0.00892	0.00808	0.00855	0.21782	0.02101	0.01907	0.02005	0.01382	0.00028	0.00024	0.00029	0.0109
MC	0		0.04044	0.04287	0.03748		0.08811	0.09342	0.08117		0.00403	0.00435	0.00369	
	5		0.05303	0.05171	0.04250		0.12375	0.11775	0.09596		0.00659	0.00636	0.00477	
	10		0.06033	0.05489	0.04266		0.14702	0.13219	0.10074		0.00841	0.00729	0.00503	



(a) Performance vs. i-bound

(b) Convergence with iterations

Figure 1.33: Random networks: KL distance

a)) averaged over all values, all variables and all problems. We also report the time taken by each algorithm. For coding networks we report Bit Error Rate (BER) computed as follows: for each approximate algorithm we pick the most likely value for each variable, take the number of disagreements with the exact input, divide by the total number of variables, and average over all the instances of the problem. We also report time.

The random networks were generated using parameters (N,K,C,P), where N is the number of variables, K is their domain size, C is the number of conditional probability tables (CPTs) and P is the number of parents in each CPT. Parents in each CPT are picked randomly and each CPT is filled randomly. In grid networks, N is a square number and each CPT is filled randomly. In each problem class, we also tested different numbers of evidence variables. The coding networks are from the class of linear block codes, where σ is the channel noise level. Note that we are limited to relatively small and sparse problem instances since our evaluation measured are based on comparing against exact

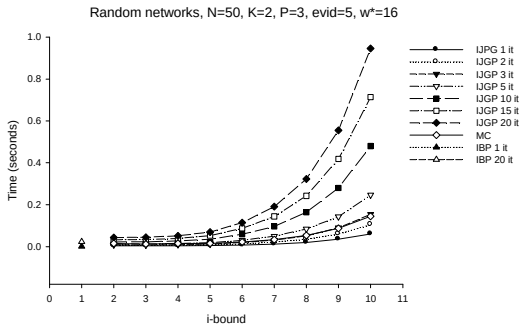


Figure 1.34: Random networks: Time

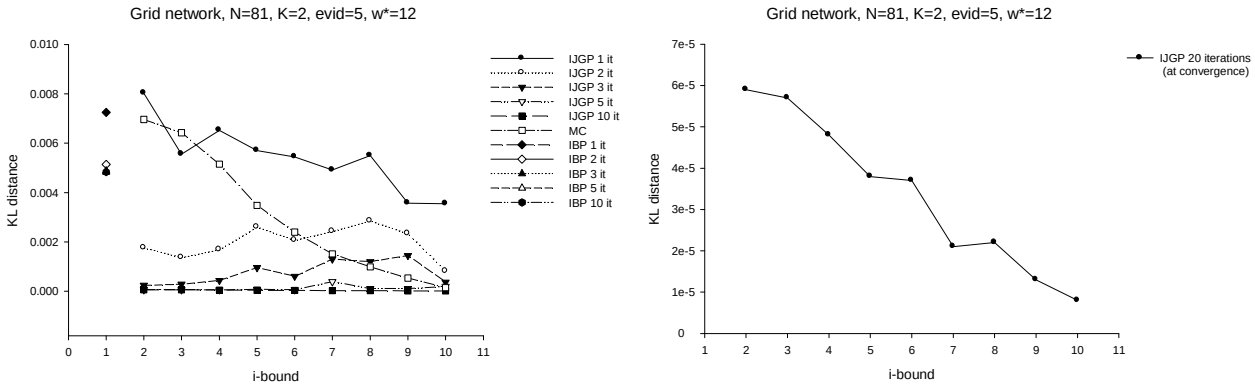
Table 1.2: 9x9 grid, K=2, 100 instances, w*=12

		Absolute error				Relative error				KL distance				
		IBP	IJGP			IBP	IJGP			IBP	IJGP			IBP
#it	#evid		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8	
1	0	0.03524	0.05550	0.04292	0.03318	0.08075	0.13533	0.10252	0.07904	0.00289	0.00859	0.00602	0.00454	0.0010
	5	0.05375	0.05284	0.04012	0.03661	0.16380	0.13225	0.09889	0.09116	0.00725	0.00802	0.00570	0.00549	0.0016
	10	0.07094	0.05453	0.04304	0.03966	0.23624	0.14588	0.12492	0.12202	0.01232	0.00905	0.00681	0.00653	0.0013
5	0	0.00358	0.00393	0.00325	0.00284	0.00775	0.00849	0.00702	0.00634	0.00005	0.00006	0.00007	0.00010	0.0049
	5	0.03224	0.00379	0.00319	0.00296	0.11299	0.00844	0.00710	0.00669	0.00483	0.00006	0.00007	0.00010	0.0053
	10	0.05503	0.00364	0.00316	0.00314	0.19403	0.00841	0.00756	0.01313	0.00994	0.00006	0.00009	0.00019	0.0036
10	0	0.00352	0.00352	0.00232	0.00136	0.00760	0.00760	0.00502	0.00293	0.00005	0.00005	0.00003	0.00001	0.0090
	5	0.03222	0.00357	0.00248	0.00149	0.11295	0.00796	0.00549	0.00330	0.00483	0.00005	0.00003	0.00002	0.0096
	10	0.05503	0.00347	0.00239	0.00141	0.19401	0.00804	0.00556	0.00328	0.00994	0.00005	0.00003	0.00001	0.0090
MC	0		0.05827	0.04036	0.01579		0.13204	0.08833	0.03440		0.00650	0.00387	0.00105	
	5		0.05973	0.03692	0.01355		0.13831	0.08213	0.03001		0.00696	0.00348	0.00099	
	10		0.05866	0.03416	0.01075		0.14120	0.07791	0.02488		0.00694	0.00326	0.00075	

figures.

Random network results with networks of N=50, K=2, C=45 and P=3 are given in Table 1.1 and Figures 1.33 and 1.34. For IJGP(i) and MC(i) we report 3 different values of i-bound: 2, 5, 8; for IBP and IJGP(i) we report 3 different values of number of iterations: 1, 5, 10; for all algorithms we report 3 different values of number of evidence: 0, 5, 10. We notice that IJGP(i) is always better than IBP (except when i=2 and number of iterations is 1), sometimes as much as an order of magnitude, in terms of absolute and relative error and KL distance. IBP rarely changes after 5 iterations, whereas IJGP(i) solution can be improved up to 15-20 iterations. As we predicted, IJGP(i) is about equal to MC(i) in terms of accuracy for one iteration. But IJGP(i) improves as the number of iterations increases, and is eventually better than MC(i) by as much as an order of magnitude, although it clearly takes more time when the i-bound is large.

Figure 1.33a shows a comparison of all algorithms with different numbers of iterations,



a) Performance vs. i-bound b) Fine granularity for KL

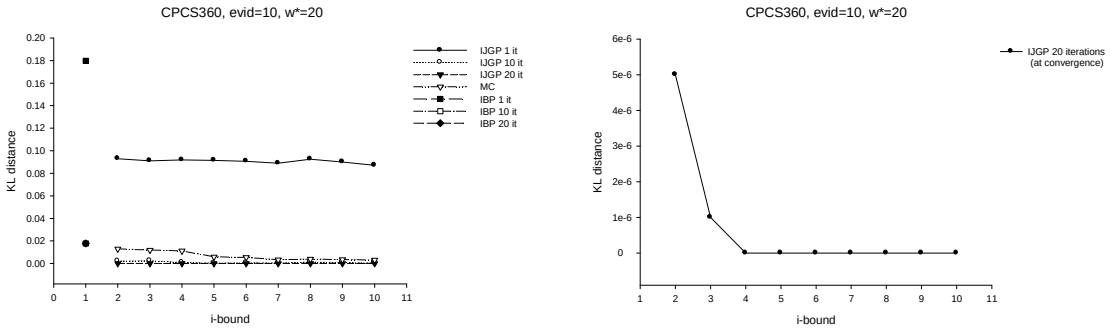
Figure 1.35: Grid 9x9: KL distance

using the KL distance. Because the network structure changes with different i-bounds, we do not see monotonic improvement of IJGP with i-bound for a given number of iterations (as is the case with MC). Figure 1.33b shows how IJGP converges with iteration to smaller KL distance than IBP. As expected, the time taken by IJGP (and MC) varies exponentially with the i-bound (see Figure 1.34).

Grid network results with networks of N=81, K=2, 100 instances are very similar to those of random networks. They are reported in Table 1.2 and in Figure 1.35, where we can see the impact of having evidence (0 and 5 evidence variables) on the algorithms. IJGP at convergence gives the best performance in both cases, while IBP's performance deteriorates with more evidence and is surpassed by MC with i-bound 5 or larger.

CPCS network results with CPCS54 and CPCS360 are given in Table **??** and Figure 1.36, and are even more pronounced than those of random and grid networks. When evidence is added, IJGP(i) is more accurate than MC(i), which is more accurate than IBP, as can be seen in Figure 1.36a.

Coding network results are given in Table 1.3. We tested on large networks of 400 variables, with treewidth w*=43, with IJGP and IBP set to run 30 iterations (this is more than enough to ensure convergence). IBP is known to be very accurate for this class of problems and it is indeed better than MC. It is remarkable however that IJGP converges to smaller BER than IBP even for small values of the i-bound. Both the coding network and CPCS360 show the scalability of IJGP for large size problems. Notice that here the anytime behavior of IJGP is not clear.



a) Performance vs. i-bound b) Fine granularity for KL

Figure 1.36: CPCS360: KL distance

Table 1.3: Coding networks: N=400, P=4, 500 instances, 30 iterations, w*=43

Bit Error Rate							
	i-bound						
σ		2	4	6	8	10	IBP
0.22	IJGP	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005
	MC	0.00501	0.00800	0.00586	0.00462	0.00392	
0.28	IJGP	0.00062	0.00062	0.00062	0.00062	0.00062	0.00064
	MC	0.02170	0.02968	0.02492	0.02048	0.01840	
0.32	IJGP	0.00238	0.00238	0.00238	0.00238	0.00238	0.00242
	MC	0.04018	0.05004	0.04480	0.03878	0.03558	
0.40	IJGP	0.01202	0.01188	0.01194	0.01210	0.01192	0.01220
	MC	0.08726	0.09762	0.09272	0.08766	0.08334	
0.51	IJGP	0.07664	0.07498	0.07524	0.07578	0.07554	0.07816
	MC	0.15396	0.16048	0.15710	0.15452	0.15180	
0.65	IJGP	0.19070	0.19056	0.19016	0.19030	0.19056	0.19142
	MC	0.21890	0.22056	0.21928	0.21904	0.21830	
Time							
	IJGP	0.36262	0.41695	0.86213	2.62307	9.23610	0.019752
	MC	0.25281	0.21816	0.31094	0.74851	2.33257	

1.10.5 Summary

In this section we presented an iterative anytime approximation algorithm called Iterative Join-Graph Propagation (IJGP(i)), that applies the message passing algorithm of join-tree clustering to join-graphs rather than join-trees, iteratively. The algorithm borrows the iterative feature from Iterative Belief Propagation (IBP) on one hand and is inspired by the anytime virtues of mini-clustering MC(i) on the other.

The empirical results are extremely encouraging. We experimented with randomly generated networks, grid-like networks, medical diagnosis CPCS networks and coding networks. We showed that IJGP is almost always superior to both IBP and MC(i) and is sometimes more accurate by an order of several magnitudes. One should note that IBP cannot be improved with more time, while MC(i) requires a large i-bound for many hard and large networks to achieve reasonable accuracy. There is no question that the iterative

application of IJGP is instrumental to its success. In fact, IJGP(2) in isolation appears to be the most cost effective variant.

Bibliography

[1] 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.

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

[3] 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.

[4] G. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding: turbo codes. In *Proc. 1993 International Conf. Comm. (Geneva, May 1993)*, pages 1064–1070, 1993.

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

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

[7] 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.

[8] M. Boddy and T.L. Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 979–984, 1989.

[9] C. Boutilier. Context-specific independence in bayesian networks. In *Uncertainty in Artificial Intelligence (UAI'96)*, pages 115–123, 1996.

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

[11] J.-F. Cheng. *Iterative decoding.* PhD Thesis, 1997.

[12] G.F. Cooper. Nestor: A computer-based medical diagnosis aid that integrates causal and probabilistic knowledge. Technical report, Computer Science department, Stanford University, Palo-Alto, California, 1984.

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

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

[15] T.L. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, 1988.

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

[17] R. Dechter. Topological parameters for time-space tradeoffs. In *Uncertainty in Artificial Intelligence (UAI'96)*, pages 220–227, 1996.

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

[19] R. Dechter. Mini-buckets: A general scheme for generating approximations in automated reasoning. In *Proc. Fifteenth International Joint Conference of Artificial Intelligence (IJCAI-97), Japan*, pages 1297–1302, 1997.

[20] 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.

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

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

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

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

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

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

[27] R. Dechter and I. Rish. A scheme for approximating probabilistic inference. In *Proc. Thirteenth Conf. on Uncertainty in Artificial Intelligence (UAI97)*, 1997.

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

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

[30] D. Draper. Localized partial evaluation of belief networks. Technical report, Phd thesis, University of Washington, 1995.

[31] F.Jensen and S.K.Andersen. Approximations in Bayesian belief universes for knowledge-based systems. In *Proc. Sixth Conf. on Uncertainty in Artificial Intelligence*, 1990.

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

[33] 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.

[34] D. Geiger, T. Verma, and J. Pearl. Identifying independence in bayesian networks. *Networks*, 20:507–534, 1990.

[35] D. Heckerman and J. Breese. Causal independence for probability assessment and inference using Bayesian networks. Technical Report MSR-TR-94-08, Microsoft Research, 1995.

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

[37] W.T. Freeman J. S. Yedidia and Y. Weiss. Generalized belief propagation. In *Advances in Neural Information Processing Systems 13*, 2001.

[38] F.V. Jensen, S.L Lauritzen, and K.G. Olesen. Bayesian updating in causal probabilistic networks by local computation. *Computational Statistics Quarterly*, 4:269–282, 1990.

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

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

[41] K. Kask and R. Dechter. Stochastic local search for bayesian networks. In *Workshop on AI and Statistics (AISTAT'99)*, pages 113–122, 1999.

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

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

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

[45] U. Kjaerulff. Reduction of computational complexity in Bayesian networks through removal of week dependencies. In *Proc. Tenth Conf. on Uncertainty in Artificial Intelligence*, 1994.

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

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

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

[49] 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.

[50] 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.

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

[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] R.A. Miller, F.E. Masarie, and J. Myers. Quick medical reference (QMR) for diagnostic assistance. *Medical Computing*, 3(5):34 – 38, 1986.

[54] R.A. Miller, H.E. Pople, and et al. Internist-1: An experimental computer-based diagnostic consultant for general internal medicine. *New England Journal of Medicine*, 307:468 – 476, 1982.

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

[56] R. Qi N. L. Zhang and D. Poole. A computational theory of decision networks. *International Journal of Approximate Reasoning*, pages 83–158, 1994.

[57] R. Parker and R. Miller. Using causal knowledge to create simulated patient cases: the CPCS project as an extension of INTERNIST-1. In *Proc. 11th Symp. Comp. Appl. in Medical Care*, pages 473 – 480, 1987.

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

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

[60] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Mateo, California, 1988.

[61] Y. Peng and J.A. Reggia. Plausability of diagnostic hypothesis. In *National Conference on Artificial Intelligence (AAAI'86)*, pages 140–145, 1986.

[62] Y. Peng and J.A. Reggia. A connectionist model for diagnostic problem solving. *IEEE Transactions on Systems, Man and Cybernetics*, 1989.

[63] D. Poole. Probabilistic partial evaluation: Exploiting structure in probabilistic inference. In *IJCAI-97: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[64] M. Pradhan, G. Provan, B. Middleton, and M. Henrion. Knowledge engineering for large belief networks. In *Proc. Tenth Conf. on Uncertainty in Artificial Intelligence*, 1994.

[65] 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.

[66] R. Mateescu R. Dechter and K. Kask. Iterative join-graph propagation. In *Uncertainty in Artificial Intelligence (UAI02)*, pages 128–138, 2002.

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

[68] 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.

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

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

[71] E. Santos. On the generation of alternative explanations with implications for belief revision. In *Uncertainty in Artificial Intelligence (UAI'91)*, pages 339–347, 1991.

[72] E. Santos, S.E. Shimony, and E. Williams. Hybrid algorithms for approximate belief updating in bayes nets. *International Journal of Approximate Reasoning*, in press.

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

[74] R. D. Shachter. An ordered examination of influence diagrams. *Networks*, 20:535–563, 1990.

[75] G. R. Shafer and P.P. Shenoy. Probability propagation. *Anals of Mathematics and Artificial Intelligence*, 2:327–352, 1990.

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

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

[78] S.E. Shimony and E. Charniak. A new algorithm for finding map assignments to belief networks. In *P. Bonissone, M. Henrion, L. Kanal, and J. Lemmer (Eds.), Uncertainty in Artificial Intelligence*, volume 6, pages 185–193, 1991.

[79] 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.

[80] M. Shwe, B.F. Middleton, D.E. Heckerman, M. Henrion, E.J. Horvitz, H. Lehmann, and G.F. Cooper. Probabilistic diagnosis using a reformulation of the Internist-1/QMR knowledge base: I. The probabilistic model and inference algorithms. *Methods of Information in Medicine*, 30:241 – 255, 1991.

[81] 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.

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

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